Korean Laid-Open Patent No. 10-2000-0069948

Job No.: O-03282
Ref.: KR2000-0069948/PU030107/JDH(SUSAN)/ORDER NOS. ART620 AND ART621
Translated from Korean by the McElroy Translation Company
800-531-9977 customerservice@mcelroytranslation.

Korean Laid-Open Patent No. 10-2000-0069948 (200.11.25) 1 copy

(19) KOREAN INTELLECTUAL PROPERTY OFFICE (KR)
(12) LAID-OPEN PATENT GAZETTE (A)

(51) Int. Cl.[6]:
G 06 F 9/44

(11) Laid-open No. 2002-0069948
(43) Publication Date: November 25, 2000

(21) Filing No.:	10-1999-7006161	
(22) Filing Date:	July 7, 1999	
Translation Submission Date:	July 7, 1999	
(86) International Filling No.:	PCT/IB1997/00025	(87) International Laid-Open No.: WO 1998/32073
(86) International Filling Date:	January 17, 1997	(87) International Laid-Open Date: July 23, 1998
(81) Designated country:	Eurpoean patents: Austria, Belgium, Switzerland, Germany, Denmark, Spain, France, England, Greece, Ireland, Italy, Luxembourg, Monaco, Netherlands, Portugal, Sweden, and Finland	
Domestic patents:	Japan, Republic of Korea, and United States of America	

(71) Applicant: Jeffrey L. Forman, International Business Machines Corporation
Armonk, New York 10504, USA

(72) Inventors: Joerg Bischof
124 In der Fadmatt, CH-8902, Switzerland

Thomas Eirich
16 Zopfstrasse, CH-8804, Switzerland

Dirk Husemann
4 Krebsbachweg, Adliswil, CH-8134, Switzerland

(74) Agents: Chang-Sei Kim
Won-Jun Kim
Seong-Gu Jang

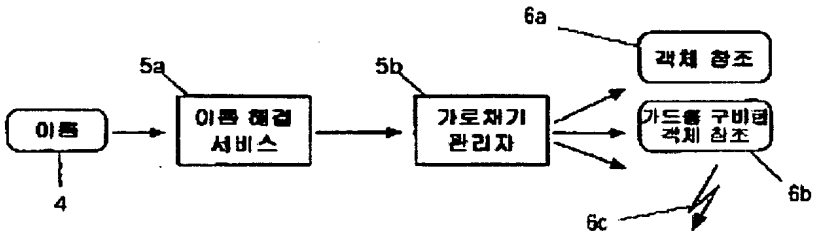
Examination Request: Filed

(54) ACCESS CONTROL DEVICE AND RESOURCE PROTECTION METHOD IN DISTRIBUTED NETWORK SYSTEM

(57) Abstract

The present invention relates to a resource protection method in an open digital system in entities or processed by entities existing in a system such as the Internet in which resources are physically and organically dispersed and independent entities are connected. In this open system, since in principle an entity has access to any resource of any of the other entities, caution is required to maintain privacy and, if necessary, confidentiality by avoiding or minimizing a contamination and breakdown risk of the resources. Therefore, the protection of the resources is very important to assure the integrity and functions of the entities. For intelligent protection of these resources, especially for the protection from access without permission, an attachable and detachable guard, which accompanies variable-grained control and/or calls of files and/or services of a namespace (consisting of all names that are provided by the entities) is provided.

Representative figure:



Key: 4 Name
5a Name resolution service
5b Interception manager
6a Object reference
6b Object reference with a guard

Specification

Technical field

The present invention relates to a method for protecting resources in entities or processed by entities in an open digital communication system that accesses these physically and organically distributed independent entities. In this open system, since in principle any entity has access to resources of other entities, it is very important to protect the resources so as to ensure the integrity and functions of the entities. As an example of this system, currently, there is the Internet provided with distributed stations, servers, and terminal user computers that provides or possesses resources such as data or program files. The present invention intelligently protects these resources, especially protects the resources from access without permission.

Prior art

Sharing of resources is a concept well known in many fields; however, the burden imposed on the maintenance of these resources can be distributed between groups by sharing rare, expensive, or infrequently-used resources. Along with the desire to share the resources, since the desire to protect the resources from use without permission has increased, the resource sharing usually means the resource protection. It is not surprising that the resource sharing and the resource protection are core issues in distributed network computers. As computers become more available and interconnected, the necessity of a means to protect the resources in these connected systems is further increased. This is requirement is crucial, especially for open networks such as the Internet. In this open distributed network system, it is essential and imperative to protect resources such as password files and individual data, or hardware resources such as file system space and communication capacity, or services such as cipher services from access without permission. Here, the protection means both basic access to the resources themselves and the use

of the resources by accessed entities. In other words, the protection means both coarse-grained resource protection (for example, an entity can use a file system service) and fine-grained resource protection (for example, an entity can use a file system service but does not have access to a password file).

The current access scheme to protect the resources in the distributed system depends upon an authentication-based technique that in most cases requires somewhat extensive changes in service-providing entities. For example, as described under the title of "Kerberos: An authentication service for open network systems" written by J.G. Steiner, B.C. Neuman, and J.I. Schiller in a pamphlet of the Usenix Conference Proceedings, February 1988, Kerberos requires an encrypted ticket that is transmitted along with each service demand. A user receives authentication through a service by a special trusted ticket and can request a Kerberos ticket for other optional services to be called upon, which are designated so that the user may use the services. Although this system is safe, it is complicated, time-consuming, and increases traffic in achieving the intended purpose of the resource protection. More specifically, the Kerberos access scheme for resource protection is not transparent for a service-providing entity (in terms of visibility), and the calling entity itself must be visually checked even for a valid ticket from the called entity, which is a drawback of a discretion-type mechanism. Codes for implementing the entity must be provided with a call for each library, and an entity that forgets to examine the ticket validity may allow any caller to access the resource. In addition, once an attacker exposes a central ticket permission service, the attacker also has access to all the entities participating in the related distribution system.

D.B. Clifton presented another solution for a resource protection task in U.S. Patent No. 5,469,556 under the title "Resource access security system for controlling access to resources of a data processing system" patented in November 1995. Clifton introduced special objects of so-called descriptors for addressing resources such as memory address in hardware. Table sets are attached to each descriptor and control the access to specific resources. Essentially, these descriptors form visual addresses of resources and realize a fine-grained access control for the resources in a system (see Clifton patent, column 3, lines 34-37)

Although the Clifton access scheme for resource protection does not show central vulnerability, unlike Kerberos, its resistance might be one of its drawbacks. The Clifton access scheme cannot be applied to a memory address translation system, that is, a distributed system bound to a single address space, in spite of the use of the descriptors. However, the distributed system, according to an actual definition, includes several address spaces connected to a kind of communication medium. In addition, since the Clifton solution is not transparent and users must use special descriptors, programs of the users must be coded accordingly.

Moreover, both the Kerberos and Clifton access schemes cannot process dynamically appearing resources during the run time of a system (and after a resource protection mechanism is set). In an environment in which an entity can enter a distributed system from the outside (for example, a Java applet entering from the Internet), dynamically generated resources exist, and these resources are also required to be protected.

Therefore, the main objective of the present invention is to realize a resource protection mechanism applicable to a dynamic distributed system. Another objective of the present invention is to generate a transparent resource protection mechanism for an entity that wants to access a program and/or a resource to be protected. In addition, another objective of the present invention is to provide a dynamically expandable resource protection method for protecting newly generated resources during the run time.

Summary of the invention

Resources to be protected from access without permission in a distributed system are usually possessed or provided by an entity such as a server or other processing stations. In principle, all entities are permitted to have access to optional resources of other entities in an open distributed system; however, as mentioned above, a security measure that protects prescribed resources and permits the access to the resources by only prescribed entities and/or under prescribed conditions is required.

In the resources, there are disk space, files, network access, etc.; however, objects and methods provided by the objects themselves can also be included. In the distributed system, the access to these resources occurs through services or methods that are provided by the entities or objects. The concept of the present invention starts from an idea for reducing the access to resources or services. To call or use a service, both the name of an entity that possesses or provides the service and the name of the service itself must be known. In actuality, for the access to the service, its name is made to correspond to its position. Namely, the name is resolved. All the names that the entities know form a namespace of the entities, and similarly, all the entity positions of the system constitute an entity position space of the system.

Therefore the underlying concept in the present invention, for example, controls a name resolution process, like the limitation of the visibility of services for entities, to control the access of the entities to the services. Another feature of the present invention is that this control is implemented, for example, by an intelligent interception manager to provide a variable-grained control of a namespace and a name resolution process. Another main feature of the present invention is to provide an attachable and detachable guard, and this guard is combined with a name, called before or after the name is used, and provided to protect fine-grained resources.

Utilizing one or more features mentioned above, a protection domain that can be dynamically constituted is generated, and the guard domain means a controlled limited environment, that is, a sandbox in which an object that can be dynamically constituted can be freely operated. An arbitrary inappropriate action effect is limited to the sandbox and cannot be propagated to the rest of a system. A reduction-oriented controlled entry, an escape point from/to the sandbox, and especially, finely adjusted shape and size can also be dynamically provided.

Brief description of the figures

Figure 1 shows an entity provided with interfaces and resources (the terms will be defined below).

Figures 2a and 2b show a method for mapping a name and its object reference.

Figures 3a-3c shows steps and modifications of the method of the present invention.

Figure 4 is a block diagram showing the embodiment of the present invention.

Detailed explanation of the invention

Next, a general overview (A) of the features of the present invention will be explained first, and then an embodiment (B) of the present invention will be explained. Both of these are examples for an object-oriented distributed network system and are processed in a Java-based network environment. Java is a programming language and environment developed by Sun Microsystems Co., located at 2550 Garcia Ave., Mountain View, CA 94043-1100, USA, and the term of Java is a trademark of this company.

A. Overview of the principle

Java environment:

Next, several common terms that are related to a Java computing environment and used in the present invention will be listed and briefly explained.

An "applet" is a Java bytecode typically loaded from a remote source and is usually not considered trusted.

An "application" is a Java bytecode loaded from a local source and is usually considered trusted.

The "Internet" is a worldwide open computer network consisting of multiple different subnetworks.

The "World Wide Web (WWW)" is an open subnetwork of the Internet.

An "intranet" is a closed internal network and is often used as a subnetwork of the Internet within a company.

A "browser" is a computer software or environment for supporting browsing (surfing) in the World Wide Web and downloading and running applets.

"Java bytecode" is a code generated by a compiler for JVM which will be mentioned below. Java Virtual Machine (JVM) is a virtual, though well-defined, processor for running the Java bytecode. It is called virtual because the machine is realized with software.

A "library" is a set of defined functions or classes that are used in a program.

A "sandbox" is a playground in which the Java applets are confined. It is a runtime environment delineated by a boundary with defined entry and exit points (gates) (for its detailed explanation, see the following content).

Sandbox security model:

The so-called sandbox described by J.S. Fritzinger and M. Mueller of the aforementioned Sun Microsystems Co. provides a playground delineated by boundaries and gates defined for the Java applet. All the applets are confined in this sandbox. To access something that is outside the sandbox, the applet must pass through the corresponding gate. Similarly, any communication with the applet must pass through a gate that can control and limit the passage. Typically, the applets are developed remotely. Typically, a user starts the downloading of an applet, and the applet usually travels on an untrusted public network such as the Internet.

Whether or not the applet is trusted depends substantially on the following elements:

- whether or not an applet programmer cannot prepare an antagonistic program, that is, whether or not a programmer is trustworthy;
- whether or not a complier that generates a bytecode according to the rules defined in a language specification is trusted;
- whether or not a network is trusted because an intruder can damage or modify the integrity of an applet.

An applet that is evaluated negatively with regard to any of the aforementioned items, is not considered to be trustworthy.

On the other hand, independent applications are usually developed locally, that is, in a safe environment or they travel on a trusted network such as an intranet. Therefore, the application is usually considered trustworthy. However, it is insufficient to discriminate the applet and the application by only the black and white logic. Since the sandbox-type access scheme is applied only to the applet and is not applied to the application, the implemented security is limited to the applet. In addition, since the currently implemented sandbox is insufficient and incomplete, multiple defects are found.

Usually, the sandbox is implemented in a browser. Since it is hard-coded without a interaction with the end user, it cannot be programmed. Therefore, it is a static model having a

fixed boundary and a fixed gate. The gate itself, that is, its position, its policy, or its existence cannot be changed, and only the decision of the gate can be changed. This model does not consider the origin of a downloaded code but generally discerns only whether or not the code is a local or remote program, that is, whether the code is the application or the applet.

For example, the Java security model already implemented cannot separate the implementation of an appropriate security policy from a mechanism for implementing the policy. Since decisions that can be controlled by most users are hard-coded in the Java virtual machine, the known security model is static, and the implementation of a sandbox paradigm is insufficient. Surely, there is a limitation in receiving and applying this known model because a security feature cannot be constituted.

The new security mechanism for an object-oriented system of the present invention can be directly applied to the Java environment. In this chapter, some of essential terms have well-accepted definitions, whereas some of them are used in a specific manner in this explanation. For the consistence of the term use, these terms will be briefly defined below.

Terminology:

"Objects" includes data and provides a method, and the data can be accessed only by this method. Objects are dynamically generated. Namely, they are momentary.

"Classes" are definitions for the objects and are static.

"Entities" are considered as objects or reference classes, regardless of whether they are active or passive. The entity possesses or provides resources.

"Interface" consists of a set of entity data and a method of an entity in which a service is accessed and changed.

"Name" is a symbolic reference for the method of an entity, and the name is not understood, that is, interpreted by a runtime element of a runtime system.

"Object reference" is a pointer for the position of the method of an entity, and the object reference is understood, that is, interpreted by a runtime environment of the runtime system.

"Service" is considered as a method of an entity indicated through one of an object reference and a symbolic reference.

Present invention:

The objective of an original security mechanism is to protect resources from access without permission in an object-oriented system. This security mechanism can be used for audit and can monitor usages of resource utilization, as well as access control. Its design is based on the basic assumption that if the access to resources is strictly controlled, none of programs are

damaged. The following elements are essential, alone or in combination, for the security mechanism of the present invention.

Object-oriented paradigm: Data are included in the entity and can be modified only by a provided interface. The access scheme to protect the entity data is to control the access to this interface.

Lazy object binding: Each symbolic reference (or name) is replaced with the corresponding object reference, through which a service can be called, at a run time. This replacement process is intercepted by the security mechanism of the present invention.

Name resolution: To obtain an object reference for a given name (symbolic reference), a name resolution is required. In this name resolution, the name is made to correspond to the object reference. In the original mechanism of the present invention, this correspondence can be intercepted by an interception manager that controls the name resolution.

Guard object: It can be optionally attached to an object reference. This guard is called before and/or after the object reference is used. In addition, it has access to context information.

Object-oriented paradigm:

A resource is included in an entity and can be accessed only through a provided interface. Therefore, in an object-oriented system, to protect the resource from access without permission, an access control issue on such a resource can be limited to an access control problem on an interface of the related entity. The entity can also provide different resources, thus being able to provide different interfaces as shown in Figure 1. To use a specific resource, the name (and the position) of the resource must be known, and the name must be changed into an object reference for indicating the position where the corresponding method exists.

Lazy object binding:

Binding is a process for connecting required elements of an executable code. The elements are generated by a compiler and called object files. The object file is provided with symbolic references for other object files. A linker connects these object files together to replace the symbolic reference with the corresponding object reference. At that time, the executable code possesses an understandable, that is, interpretable, necessary object reference by a runtime environment, so that the executable code can call a reference method.

If the replacement is carried out at runtime, it is called "dynamic binding," and otherwise, that is, if the replacement is carried out before the execution of the code, it is called "static binding."

If all the symbolic references are replaced at one time, the connection process is called "eager binding." An object reference for a symbolic name is carried out only once for a run time,

it is called "lazy binding." Once the replacement is carried out with executable code, it is fixed and is no longer subject to change.

For example, C++ compiler forms additional symbol information by combining a human-readable symbolically referenced method of another entity (that is, name), with its pattern information. Any of these references is still a symbol (that is, name); however, it is no longer a human-readable format. A binder includes an appropriate code in accordance with a symbolic reference and replaces the symbolic reference with an appropriate object reference. The object reference indicates the start of the position where the method exists. The Java environment uses a lazy binding access scheme. Once the runtime environment uses each symbolic reference, the symbolic reference is replaced.

In a distributed access scheme, a method through an address space and/or a system can be invoked. In most cases, a remote object is expressed by a proxy object that acts as a local representative. Usually, the proxy object searches for the remote object and generates, destroys, and invokes the remote object. Therefore, the object reference indicates the position of the corresponding proxy object.

Interception and name resolution:

The feature of the original mechanism of the present invention is that the binding process is intercepted, regardless of the eager binding or lazy binding, and an interception manager is inserted. The resolution of a symbolic reference, that is, a name requires the correspondence between the latter and an object reference. The process for finding the corresponding object reference for a given symbolic reference (name) is a name resolution process. It is implemented by the name resolution process as shown in Figures 2a and 2b. To employ a specific method, an invoking object must know a symbolic reference or name to be resolved with an appropriate object reference. If the entity does not know the symbolic reference or name, a reference and an entity, which are expressed with the symbolic reference, reliably know the symbolic reference or name, and references that are expressed by the object reference are separated, the following two difference spaces are generated. First, a "presumed namespace (concrete namespace)" of the entities consists of a set of symbolic references. Although these symbolic names are not yet resolved, they are subjected to the resolution process. The entities presume that these references exist; however, this presumption has not yet been verified through the resolution process. Their methods can neither yet be invoked nor understood by a run time and a runtime element. Secondly, a "concrete namespace" of the entities consists of a set of object references. Since these object references are resolved names and have been verified through the resolution process, the entities clearly know them. The object references can be understood by a run time and a runtime environment.

The set of methods indicated in the "concrete namespace" is usually a subset of the methods indicated in the presumed namespace. Although the references (object references and symbolic references) are different, the methods indicated by them are the same.

Interception manager:

A so-called interception manager shown in Figure 2b is one feature of the present invention that can optionally be provided. When the interception manager is provided, its main mission is to control and correct the name resolution process according to a specific security policy. For example, the concrete namespace can be reduced by blocking the resolution of several symbolic names. If the name resolution process is intercepted and the interception manager is inserted, an access control mechanism is constituted. The regulation of the concrete namespace for the security purpose is based on the assumption that none of resources can be used without being provided with an object reference of an entity that represents the resources. A symbolically referenced method is resolved only when the interception manager permits this resolution.

As shown in Figure 2b, the interception manager is part of the name resolution process. The object reference as a result of the name resolution process is intercepted, and the interception manager delivers a notice that an unmodified object reference or an object reference attached with a guard object or an exceptionally referenced entity does not exist. The density of the access control implemented by the interception manager depends upon the density of the resolution process. More precisely, the interception manager can act at a coarse level instead of a fine level. Nevertheless, a variable-grained access control still exists. The interception manager accesses context-sensitive information such as source entity name, destination entity name, and their data. The control of only the access is insufficient in certain cases, and auditing or monitoring the use of resources may be required. This requires a further invoking of a security mechanism element before and/or after protected resources are used. This object is called a guard and can be inserted by the interception manager.

Guard object:

A guard object as another optional feature of the present invention is combined with a specific object reference, and it is called before and/or after the object reference is used, depending upon how the interception manager installs the guard object.

Before a related object reference is used, if the guard object is called and accesses an argument provided for a destination entity, it is said that the guard object has been installed before the destination entity. After a related object reference is used, if the guard object is called, the called message is delivered, and the guard object accesses the delivered value, it is said that

the guard object has been installed after the destination entity. Figure 2b shows a modified object reference attached with the guard object as one of the delivery results of an intercepted name resolution process.

If there is no error in the guard object, the implementation is continued normally. Similarly to context information used by the interception manager, the guard object also accesses context-sensitive information. Source entity name, destination entity name, and call parameters are included in the information. Before the guard object is installed before the destination entity, the following actions are carried out:

- a call is refused.
- a call is passed through without changing it.
- a call is changed into provided data.
- the destination of a call is changed.
- the rights for a caller entity are assigned and/or checked.
- the access is monitored.
- a notification and/or an audit monitoring service is carried out.

If the guard object is installed after the destination entity, the following actions are carried out:

- a delivery parameter or state is changed.
- a prior assigned right is removed.
- a response is monitored.
- a notice and/or an audit monitoring service is carried out.

The guard object is not affected by a source or destination, that is, a calling entity or an entity to be called. The reason for this is that it is attached from the outside by the interception manager.

Regarding the invoking entity or the entity to be called is concerned, the guard object cannot call it, and it is transparent. Contrary to the interception manager, the guard object cannot directly reduce the concrete namespace and can have only an indirect influence on the namespace. However, the original symbolic reference can be formed by inversely resolving the attached object reference. As a result of this inverse resolution, when this method is used next time, the symbolic reference is run into again, followed by a call for the name resolution and the interception manager. The interception manager has a possibility of re-modifying the concrete namespace of the call object according to the security policy. The inverse resolution of the object reference is useful for the case where the security policy is changed after the symbolic reference is resolved.

In addition, each guard object has a possibility of determining how long the guard object wants to be attached to the object reference. The guard object can be determined so that it is self-separated, and in this case, it is no longer invoked by the call of the object reference.

Capability:

The capability is a right to call a specific object in a specific mode (it will be explained in further detail later). In a book titled "Modern Operating System[s]" by A.S. Tanenbaum, published by Prentice Hall International Co. in 1995, the following three elements can be included.

(1) Pointer to objects: A necessary pointer is achieved by a resolved reference (object reference). If executable code is provided with the pointer, it is assumed that an appropriate runtime system can be called. This means that all the entities having a capability can call the corresponding method.

(2) Pattern information of objects: Required pattern information is achieved through an attached guard object. The pattern of the capability depends upon a related guard object.

(3) Access right for objects: A required access right is achieved by an attached guard object. It can check the right, and a calling entity must provide a right to use a specific method. In addition, if the calling entity is provided with an appropriate right, a guard object reinforces the existing right or can also assign a new right to utilize this service.

The capabilities of an object reference and a guard object are similar, but they are not the same. A resolved name provided with an installed guard object may be regarded as an improved dynamic capability. The dynamic element of the capability is a bound guard object.

Overview summary:

The access scheme proposed to protect resources in an object-oriented system includes one or more of the following features. An object-oriented paradigm well controls the access to resources through an object and its interface. Access control is achieved through the control of a namespace for a calling entity to control the visibility of a resolved name. If the calling entity cannot view a service, the entity cannot call the service and cannot use related resources without the possibility of calling a method. Since a resolution manager (and a guard object in accordance with the selection) may utilize context information, the resolution manager can act very flexibly and is dependent on contexts. Even if an entity is not known, the guard object can be attached from the outside and is invisible and transparent as far as a calling entity and an entity to be called. Therefore, the mechanism protects resources in a dynamic form at low cost in terms of performance burden.

B. Embodiment

Next, the embodiment of the security mechanism of the present invention will be described in the Java environment. All the changes made for the Java Virtual Machine (JVM) including an interpreter and a runtime library will be investigated. In addition, the elements of the security mechanism element of the present invention realized in Java will also be presented.

In this chapter, since the detailed contents of the embodiment are handled, it is assumed that any party concerned is familiar with the basic mechanism of the embodiment in JVM and C as well as the Java programming language itself. The comprehensive presentation on the Java language and the embodiment in JVM are described in a book titled "The Java Language Specification" by J. Goslin and G. Tele and a book titled "The Java Virtual Machine Specification" by T. Lindholm and F. Yellin published by Addison-Wesley Publishing Co. in 1996. The embodiment elements of the security mechanism of the present invention are based on Java Development Kit (JDK) version 1.0.2 and the AIX version of JVM. This chapter describes the elements of protection system resources expressed with a system class. Though a slight change in the Java virtual machine was inevitable, the change was maintained to be as minimal as possible.

B1. Function overview

The embodied function of the security mechanism of the present invention includes the embodiment of an intercept manager and several basic guard objects. The term of the entity has been explained in detail in the previous chapter to apply the entity to a specific class. The class can be regarded as an entity because it possesses and indicates resources. Since system resources are expressed through the corresponding system classes in the Java library, the class is not ambiguous but is known as a sufficiently officially recognized name. Therefore, this embodiment protects the system classes existing in the Java library. An object reference received from a name resolution process (see the following) is a pointer to the corresponding part of the code that substantially indicates the start of a method description.

The protection of the system classes is achieved through the access control by modifying the namespace. If the concrete namespace of an object includes an object reference, the subsequent access to a referred method is permitted. Therefore, as described in the above assumption, a program has access to only resources that can utilize the object reference. To control and modify the concrete namespace of the object, the lazy binding mechanism and the name resolution are utilized by inserting the intercept manager as mentioned above. The guard objects are bound with the object references. In order to call the guard object before the original method is implemented, the method call technique is expanded so that it is provided with an examination of the installed guard object. To realize this function, the following subjects:

- resource protection through a namespace,
- change of a method call element in the Java virtual machine,
- intercept information base class that accesses the original C code,
- realization of an intercept manager, and
- guard object base class

are very important.

B2. Java virtual machine change-system level

In this section, the reinforcement of the Java virtual machine and the Java runtime library will be discussed. The reinforcement is realized in the original C code. Therefore, it is machine-dependent. To realize the security mechanism of the present invention, the following elements:

- name resolution element of the virtual machine,
- all opcodes that handle a method call,
- binary method expression in memory, and
- thread structure

were changed.

Namespace modification:

Access control is achieved, and as a result, to achieve the resource protection, the concrete namespace of a called object is controlled and reduced or expanded.

To modify the namespace of a starting object, the lazy object binding method of Java is employed. The binary file layout of Java refers to other classes and interfaces and their fields, methods, and preparers symbolically by utilizing sufficiently officially recognized names. For the fields and the methods, these symbolic references include the names of class or interface patterns for declaring the fields or methods along with appropriate pattern information as well as the names of the fields or methods themselves. Before the object is usable, its name must be resolved and become an object reference. An identifier verifies whether or not the object reference is correct. Typically, if the reference is repeatedly used, the object reference is replaced with a direct object reference that can be more efficiently processed. If an error is generated, an exception occurs. For more details, refer to a book titled "The Java Language Specification" by J. Goslin, B. Joy, and G. Tele and a book titled "The Java Virtual Machine Specification" by T. Lindholm and F. Yellin published by Addison-Wesley Publishing Co. in 1996.

After the name is resolved into an object reference, the intercept manager is called to determine whether or not the resolution of a symbolic reference is allowed (see Figure 4). Therefore, the name resolution is intercepted, realizing the lazy object binding. In accordance

with the return value of the intercept manager, a concrete namespace is modified as described below.

If the intercept manager returns "false," a NoClassDefFoundError exception is provided, indicating the absence of a class file to a called object. Therefore, the name of a desired resource cannot be resolved, so the position of the corresponding part of the code is not known. This technique that inquires of the intercept manager and then provides an exception reduces the concrete namespace of the called object.

If the intercept manager returns "true," the call continues normally, and the concrete namespace is not modified. In actuality, the concrete namespace is expanded as a resolved name, that is, as a pointer to the corresponding part of the code provided with an object reference.

The following code segment C1 illustrates the new code inserted into a name resolution function. This C code segment carries out the following steps:

(1) A required structure is initialized to manage a guard object.

(2) Whether or not a required resolution is permitted is checked, and if it is not permitted, a non-resolution is indicated, and "false" is returned.

(3) Otherwise, the method is indicated by a resolution, and the symbol information is replaced with the pointer to the corresponding part of the code.

Since the intercept manager can attach several guard objects, the method call technique requires the modification as well as the name resolution process.

Code segment C1:

This code segment was inserted into a function for resolving the symbolically referred name. An appropriate exception is provided to the ic_CheckMethodResolving( ) method (in this code, the guard object is called a "filter object").

```
[Resolution method]
    /*
    If the corresponding method expression can be found, a filter delivery table for this
method is prepared.
    */
    ic_PrepareFilter (class, mb);
    /*
    Whether or not this specific name resolution is permitted is checked.
    */
    if (!ic_CheckMethodResolving(mb)) {
    /*
    If the resolution of this method is not permitted, it is indicated as a non-resolution.
```

```
        */
        fieldfiltertable(&mb->fb)->resolved = FALSE;
        return FALSE[;]
        }
        /*
        The method is indicated as a resolution.
        */
        fieldfiltertable(&mb->fb)->resolved = TRUE[;]
        [/*]
        The symbolic name information is changed to a pointer to an appropriate method block.
        */
        constant_pool[index].p = mb;
        continue_as_usual( );
[...]
```

Method call:

The Java interpreter is in charge of running a method that has already been resolved. A detailed discussion on the method call and the related procedure is described in a book titled "The Java Virtual Machine Specification," by T. Lindholm and F. Yellin, published by Addison-Wesley Publishing Co. in 1996. Figure 4 shows a brief summary of a control flow of the method call. In Figure 4, ten new elements added by the present invention are 19-25 and 27-29, and the other elements are the original Java elements. The new elements of the present invention include the check on the name resolution and the implementation of guard objects.

opcode:

opcode is a short term showing the operation of a specific set and is usually a number. It represents a specific function. For example, opcode 182 (0xb6), related to an associative memory code opc_invokevirtual, shows a basic step for calling a virtual method. A detailed explanation of various Java codes and functions which are provided by these codes can be found in the aforementioned book by T. Lindholm. To install and maintain the guard objects, the embodiment of the opcode for handling the method call is changed. The function is expanded so that a code for checking the guard objects and its subsequent execution is provided. After the last guard object is executed, the original method is executed. C2 is a C code segment inserted into all the opcodes that handle the method call. This code segment carries out the following steps:

(1) A step for checking whether or not the guard object is installed and continuing execution normally if there is no installed guard object,

(2) A step for checking whether or not the method is resolved and calling an installed guard object if the method is resolved, and

(3) A step for checking whether or not an exception is handled, stopping execution of the method if the exception is provided, and handling the exception.

To attach the guard object to an object reference, the memory layout of the original method is expanded.

Code segment C2:

This code segment is inserted before execution of the original method.

[Execution method]

```
/*
Is a filter delivery table installed?
If it is installed, whether or not a filter object is installed is checked.
*/
if (fieldfiltertable(&mb->fb) &&
        (fieldfiltertable(&mb->fb)->used_filter)) {
/*
Is this method resolved through the mechanism of the present invention?
*/
if (fieldfiltertable(&mb->fb)->resolved == TRUE) {
/*
* The installed filter object is called.
*/
ic_InvokeFilterObjects(fieldfiltertable(&mb->[sic; omission in source]), ee);
} else {
/*
The namespace is checked and an intercept manager object is called.
*/
ic_CheckMethodResolving(mb);
}
/*
Is everything all right? No exception has been provided to anyone.
*/
if (exceptionOccurred(ee))
        goto handle_exception;
}
```

continue_executing_original_method( );

[...]

Method expression:

In order to execute an arbitrary method of an object, the method requires the corresponding binary expression in memory. The binary expression of the method is divided into a static element and a dynamic element. The static element includes its class depiction and static field, and the dynamic element includes its dynamic field. A text segment and a data segment of the binary expression executable in a UNIX operating system are similar concepts, and two segments correspond to different regions of an address space. The text segment corresponds to a read-only mode, and the data segment corresponds to a read/write mode.

The static element of the method is shared by all the objects instantiated from the same class depiction; however, each object has its own dynamic element. The guard object is attached to the static element (fieldblock structure) of the method. Therefore, the guard object is also shared by all the instances of the same class description. Though the guard object is attached to the static element, the guard object is attached to a different object and its method rather than being attached to a class and its method.

Guard delivery table

A guard delivery table manages the attached guard object. It has the references of each guard object and shows how many guard objects are installed. A header consists of the following management elements:

a variable (indicated by used_filter) showing how many guard objects are used,

a variable (indicated by res_filter; however, it must always be established that res_filter is greater than used_filter) showing the number of guard objects to which a space is allocated,

a flag (indicated by resolved) showing whether or not it is considered that the method is resolved,

a pointer (indicated by mb) to an attached method expression,

a pointer (indicated by caller) to a class filter of a calling object, and

a pointer to the first element of a dynamic array including a guard object structure that is managed. (For historical reasons, the guard object is called a filter object in the code.)

The next code segment C3 illustrates the structure of the guard delivery table header.

Code segment C3

It is a header of the guard delivery table.

struct filterdtable {

```
        int used_filter;
        int res_filter;
        int resolved;
        struct methodblock *mb;
        struct ClassClass *caller;
        struct filterdesc *filters[1];
};
```

The pattern of the array element indicated by filters is illustrated in the code segment C4, and it includes the following elements:

a pointer (indicated by obj) to an object memory position of the guard object,

the method (indicated by methodname) of a method to be called,

the corresponding signature (indicated by signature) because two methods can have different signatures, though the name is the same, and

an answer call method name (indicated by notify) for designating a method in an intercept manager class called when the guard object separates itself from the corresponding object.

Code segment C4:

Each element in the filter delivery table is an instance of filterdesc.

```
struct filterdesc {
        HObject *obj;
        char *signature;
        char *methodname;
        char *notify;
};
```

Since the guard object is realized by Java, it is a latent candidate for the guard object that modifies the namespace and is attached. Therefore, to avoid the repetition problem, an executing thread indicator is attached.

Thread:

The thread is an execution path in an address space. This address space can be shared by many methods that are currently running. A detailed discussion on the thread concept is shown in a book titled "An Introduction to Programming with Threads" by A.D. Birell, described in the System Research Center (SRC) Report from Digital Equipment Corporation issued on January 6, 1989.

The guard objects and the intercept manager of the present invention are implemented in Java. Since they also access resources, a criterion for discriminating elements belonging to the mechanism of the present invention from elements that do not belong to the mechanism is required. Otherwise, each time the intercept manager accesses resources and calls the method, the intercept manager itself checks the elements. This repetition can disappear if the thread is appropriately indicated. If the thread enters the guard object or the intercept manager, it is indicated that the thread is in a supervising state. Next, if the thread makes an attempt to enter the guard object or intercept manager, whether or not the thread is indicated in a supervising state is checked. In accordance with the existence of this indication, the system determines whether or not the thread is applicable to a new mechanism. If the guard object or intercept manager leaves, the pointer is erased.

This temporarily allocated state is similar to the system call concept of traditional UNIX, in which the process that invokes a system call has more rights to access resources than the process that does not invoke a system call, to some degree. In kernel mode, a supervising state is temporarily allocated to an invoking process. With this privilege, the process accesses protected resources possessed by the operating system. When the process leaves the system call, the privileged state is erased. A detailed explanation of the system call concept is described in a book titled "4.4 BSD UNIX Operating System" by S.J. Leffler et al., published by Addison-Wesley Publishing Co. in 1996.

The guard objects and the intercept manager also have access to context data, including arguments provided by the original method, and modify the arguments. If the call of the original method is intercepted and the intercept manager is requested or the guard object is called, the invoking thread is temporarily allocated a pointer to the stack frame of the original method. Therefore, the pointer accesses the method argument situated on the stack.

When entering the security element according to the present invention, a pointer to an appropriate guard delivery table is allocated to the invoking thread. Therefore, the thread completely accesses all the guard objects.

In order to achieve the aforementioned function, the thread structure is expanded so that it is provided with the following elements:

- a single bit (indicated by sv_thread) showing whether or not the thread is in a supervising state,

- a pointer (indicated by iv_filtertable) to an appropriate guard delivery table, and

- a pointer (indicated by ic_optop) to an appropriate stack frame of the original method.

These elements are allocated when the thread runs across the boundary between ordinary code and security code.

Java class layer-user level:

In this paragraph, elements embodied by Java will be described. Embodying a new security mechanism by Java follows one of the main principles of the Java philosophy, that is, "portable, if possible."

Therefore, most of the embodiments are implemented in Java using an interface well defined for the elements embodied in the original C code. A routine embodied with the original C code can be accessed through an abstract base class called InterceptionInfo.

Intercept information:

InterceptionInfo is an abstract class implemented in Java. It is utilized for providing a basic security function of the present invention. A discussion on a class modifiers in Java is provided by Goslin et al., cited above. The elements of this base class can be divided into the following categories:

- installation management of an optional guard object,
- removal management of an optional guard object,
- sequence management of an installed guard object,
- fetch and modification of an optional argument, and
- fetch of sufficiently officially recognized names, method names, and signature names of called object/entity and object/entity called.

The overview of an Application Programming Interface (API) provided by the InterceptionInfo base class is provided from the code segment C5. To obtain a sufficient security function of the present invention, the intercept manager class and all the guard objects must sub-class this InterceptionInfo base class as shown in the figure.

Code segment C5:

```
This class provides an interface for the embodied original C code.
public abstract class InterceptionInfo {
        protected native void unresolveMethod( );
        protected native int getIntArgument (int index)
                throws ArgumentAccessException;
        protected native void setIntArgument (int index, int value)
                throws ArgumentAccessException;
        protected native float getFloatArgument (int index)
                throws ArgumentAccessException;
        protected native void setFloatArgument (int index, float value)
```

```
        throws ArgumentAccessException;
protected native byte getByteArgument (int index)
        throws ArgumentAccessException;
protected native void setByteArgument (int index, byte value)
        throws ArgumentAccessException;
protected native Object getObjectArgument (int index)
        throws ArgumentAccessException;
protected native void setObjectArgument (int index, Object value)
        throws ArgumentAccessException;
protected native void printArgument (int index);
protected native Object getClassLoader( );
protected static native String getCaller( );
protected static native String getCallee( );
protected static native String getMethodName( );
protected static native String getSignature( );
protected static native boolean
        pushFilter (InterceptionFilter arg, String filtermethod,
                String filtersignature,
                String ic_manager_notify);
protected static native boolean popFilter( );
protected static native Object[ ] getFilterList( );

} // Class InterceptionInfo
```

Intercept manager class:

The intercept manager is embodied by Java and is regarded as final and static. The intercept manager sub-classes the InterceptionInfo base class. It is in charge of controlling and modifying the namespace of a calling object. The intercept manager embodies a security policy for the Java environment, regardless of whether the call entity is part of the application or part of the applet, that is, without discriminating between the application and the applet. The intercept manager defines the boundary of a security sandbox and implements a gate for communications with the outside of the sandbox.

Since Java itself manages memory, its garbage collection is responsible for removing an unused object and freeing the memory. In case no reference exists at Java level, it is regarded that the object is not in use. Therefore, the intercept manager must track and monitor all the guard objects generated and installed by the intercept manager. Otherwise, when memory is

freed, the garbage collection removes this object. A detailed discussion on garbage collection is provided from the above-cited book by Lindholm. The intercept manager generates guard objects and determines a certain guard object that is attached to a certain method. The manager also designates the sequence by which the guard objects are called before and/or after the original method.

Intercept filter:

The InterceptionFilter class provides a basic function required by a guard object. Like the intercept manager, the InterceptionFilter class is sub-classed to access the original C code method. This class expands the sub-class by a method for separating the sub-class from the original method (object reference attached to the class). In an actual embodiment, the guard object is executed before the original method is called. The code segment C6 shows a basic guard object base class. The defaultCallFilter( ) method is a method for calling an object reference as a default before it is called.

Code segment C6:

```
It expands the InterceptionInfo abstract base class.
public class InterceptionFilter extends InterceptionInfo {
        protected int unresolve;
        private protected InterceptionFilter( ) {
                unresolve = 0;
        }
        private protected void defaultCallFilter( ) {
        throw new ICFilterException ("no filter method
        implemented");
        }
        private protected native void detachMySelf( );
}
```

Conclusion:

This chapter B has handled a core subject called the embodiment of the concept expressed in the chapter A. An essential function for implementing the security mechanism of the present invention was described as a subset of the architecture proposed. This architecture addresses the principle of controlling access to a system class representing a system resource and calling an optional guard object.

The modification in the Java virtual machine was limited to the minimum. In order to modify the concrete namespace of a called object, a significant change was made in the name resolution process, and a significant change was also made in the alternative or selective name resolution process through the guard object, so that the security for all the method calls can be assured. Because of the intercept manager and all the guard objects, a programmer can constitute a variable-grained protection mechanism, which can be optimized in terms of desired protection level as well as economy.

Claims

1. A resource protection method in a distributed network system, wherein in a resource protection method for protecting resources (2a, ..., 2n), such as files or other objects, in a data processing system provided with a name resolution mechanism, especially in a distributed network system, said resources are called by a first entity, that is, a calling entity (7) and provided by a second entity in said system, that is, an entity (9) to be called;

access to said resources is called by said calling entity (7), utilizing symbolic names, in which all the symbolic names form a presumed namespace for the entities;

said name resolution process allocates object references to said symbolic names, in which all the object references form a concrete namespace for the entities that reflect permitted resources; said permitted resources are a subset of all the resources of said system; and

said calling entity (7) receives permission to access only said concrete namespace in said entity (9) to be called.

2. The resource protection method in a distributed network system as cited in Claim 1, wherein said symbolic name includes at least an entity name and a method name; and all of said names form said presumed namespace.

3. The resource protection method in a distributed network system as cited in Claim 1 or 2, wherein if said access is allowed, said name resolution process accesses the resources (2a, ..., 2n) by calling a desired method from the entity (9) to be called, resolving the symbolic name by an object reference (6) for permitting the access of the calling entity (7), especially an immediate access.

4. The resource protection method in a distributed network system as cited in any of Claims 1-3, wherein said problem resolution process is modified by providing a guard object (13) and replacing said guard object (6) with a reference to said guard object.

5. The resource protection method in a distributed network system as cited in Claim 4, wherein said guard object hides said object reference (6b) from said calling entity (7) to provide conditional access to a desired method and to control said call process.

6. The resource protection method in a distributed network system as cited in Claim 4 or 5, wherein the guard object is called before or after the object reference is used.

7. The resource protection method in a distributed network system as cited in Claim 4 or 5, wherein the conditional access provided by said guard object depends upon a more detailed analysis, compared with the fine breakdown, that is, a breakdown used in the name resolution process.

8. The resource protection method in a distributed network system as cited in any of Claims 1-7, wherein at least one step of said method is realized in an object-oriented mode.

9. An access control device in a data processing network, wherein in an access control device for resource protection including a means for protecting resources (2a, ..., 2n), such as files or other objects, from access without permission in a data processing network, said access control device, in which said resources are called by a first entity, that is, a calling entity (7) and provided by a second entity in said network, that is, an entity (9) to be called, includes

a symbolic name resolution means for resolving a symbolic name that is used in invoking the access of said resources by said calling entity (7), in which a presumed namespace is formed for the entities by all of said symbolic names;

an object reference allocation means for allocating object references to said respective symbolic names, in which a concrete namespace that reflects permitted resources is formed and said permitted resources are a subset of all resources; and

an access limitation means for limiting the access of said calling entity (7) to said concrete namespace in said entity (9) to be called.

10. The access control device in a data processing network as cited in Claim 9, wherein said device further includes a means that installs a guard object (13) and replaces said object reference (6) with a reference to said guard object.

11. The access control device in a data processing network as cited in Claim 9 or 10, wherein at least one of said means is realized by software in an object-oriented mode.

12. A data processing network wherein the method of any of Claims 1-8 and/or the name resolution system of any of Claims 9-11 are realized.

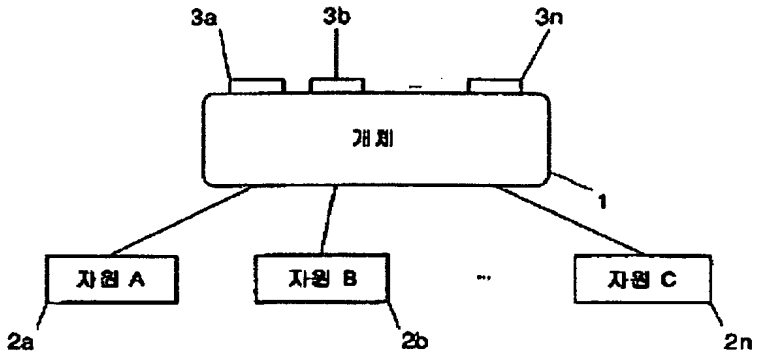


Figure 1

Key: 1 Entity
2a Resource A
2b Resource B
2n Resource C

Figure 2a

Key: 4 Name
5 Name resolution service
6 Object reference

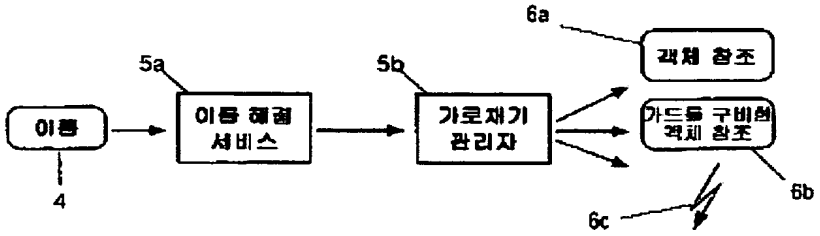


Figure 2b

Key: 4 Name
5a Name resolution service
5b Intercept manager
6a Object reference
6b Object reference with a guard

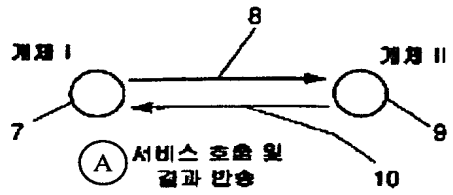


Figure 3a

Key: A Service call and result delivery
7 Entity I
9 Entity II

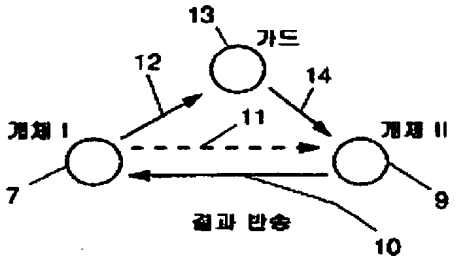


Figure 3b

Key: 7 Entity I
9 Entity II
10 Result delivery
13 Guard

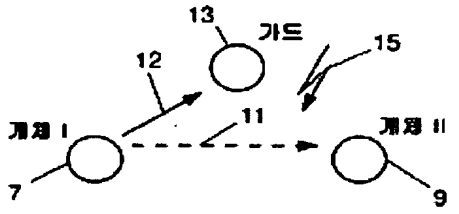


Figure 3c

Key: 7 Entity I
9 Entity II
13 Guard

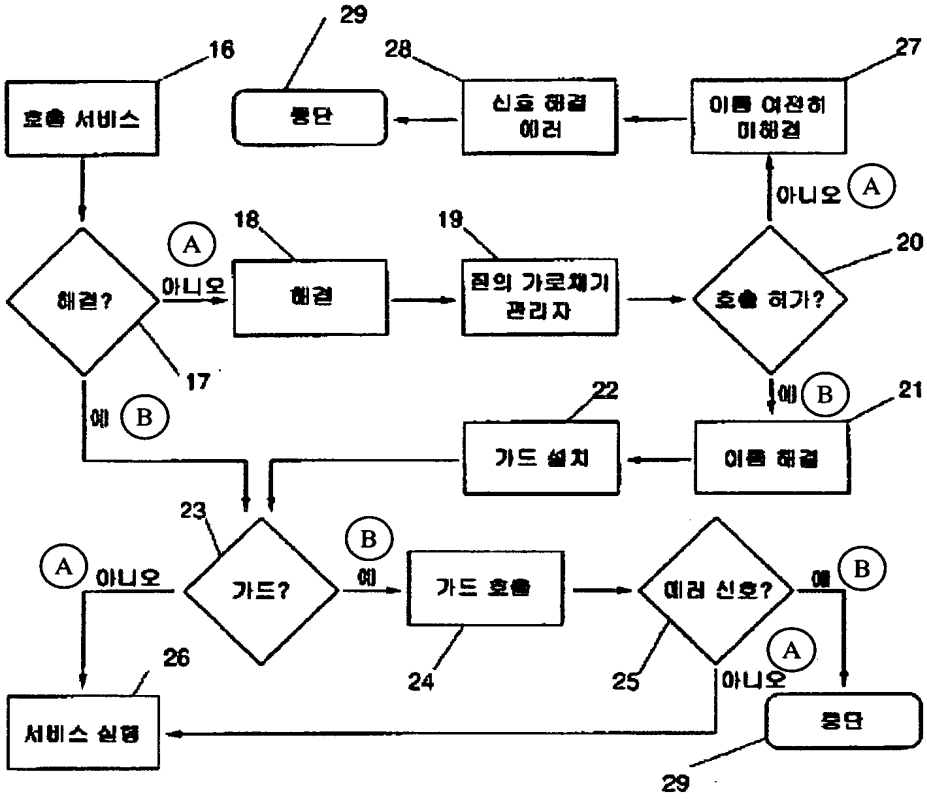


Figure 4

Key:

A	No
B	Yes
16	Call service
17	Resolution?
18	Resolving
19	Inquiry intercept manager
20	Call permission?
21	Name resolution
22	Guard installation
23	Guard?
24	Guard call
25	Error signal?
26	Service implementation
27	Still name unresolved
28	Signal resolution error
29	Stop

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.[6]
G06F 9/44

(11) 공개번호 특2000-0069948
(43) 공개일자 2000년11월25일

(21) 출원번호	10-1999-7006161		
(22) 출원일자	1999년07월07일		
번역문제출일자	1999년07월07일		
(86) 국제출원번호	PCT/IB1997/00025	(87) 국제공개번호	WO 1998/32073
(86) 국제출원출원일자	1997년01월17일	(87) 국제공개일자	1998년07월23일
(81) 지정국	EP 유럽특허 : 오스트리아 벨기에 스위스 독일 덴마크 스페인 프랑스 영국 그리스 아일랜드 이탈리아 룩셈부르크 모나코 네덜란드 포르투칼 스웨덴 핀랜드 국내특허 : 일본 대한민국 미국		
(71) 출원인	인터내셔널 비지네스 머신즈 코포레이션 포만 제프리 엘 미국 10504 뉴욕주 아몬크		
(72) 발명자	비스콤조에르그 스위스체하-8902어르도르프인데르파드마트124 에이리치토마스 스위스체하-8804에이유조프스트라세16 후세만디크 스위스체하-8134아드리스윌크렙스배치웨그4		
(74) 대리인	김창세, 김원준, 장성구		

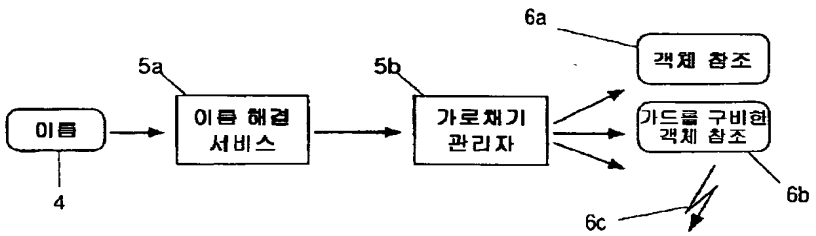
심사청구 : 있음

(54) 분산 네트워크 시스템에서의 접근 제어 장치 및 자원 보호 방법

요약

본 발명은 자원이 물리적 및 유기적으로 분산되고 독립적인 개체들이 연결된 인터넷과 같은 시스템에 존재하는 개체 내에 있거나 개체가 소유한 개방형 디지털 시스템 내에서의 자원 보호 방법에 관한 것이다. 이와 같은 개방형 시스템에서는 원칙적으로 한 개체가 다른 어떤 개체의 어떤 자원에도 접근할 수 있기 때문에, 자원의 오염 및 파괴 위험을 피하거나 최소화하여 프라이버시 및 필요할 경우 비밀을 유지하기 위해 세심한 주의를 기울여야 한다. 따라서 자원의 보호는 개체의 통합성 및 기능을 보증하기 위해 극히 중요하다. 이와 같은 자원의 지능적인 보호를 위해, 특히 무단 접근으로부터의 보호를 위해, (개체가 제공하는 모든 이름으로 구성된) 이름 공간의 가변 결정립 제어 및/또는 파일에 대한 각 호 및/또는 서비스에 수반하는 탈부착이 가능한 가드가 제공된다.

대표도



명세서

기술분야

본 발명은 물리적으로 뿐만 아니라 유기적으로 분산된, 독립적인 개체를 접속하는 개방형 디지털 통신 시스템에 있어서, 이들 개체 내에 있거나 객체가 소유한 자원의 보호 방법에 관한 것이다. 이와 같은 개방형 시스템에서는 원칙상 어떠한 객체도 다른 개체의 자원에 접근할 수 있기 때문에, 자원을 보호하는 것은 개체의 통합 및 기능을 보장하는 데 있어서 극히 중요하다. 이와 같은 시스템의 한 예로서는, 현재 예를 들어 데이터나 프로그램 파일과 같은 자원을 제공하거나 소유하는, 분산된 스테이션과 서버와 단말 사용자용 컴퓨터를 구비하는 인터넷이 있다. 본 발명은 이와 같은 자원의 지능적인 보호, 특히 무단 접근으로부터 자원을 보호하는 것에 관한 것이다.

배경기술

자원 공유는 많은 분야에서 잘 알려진 개념인데, 드물고 비싼 또는 그다지 사용되지 않는 자원을 공유함으로써 이 자원을 유지하는 데 드는 부담을 사용자 그룹 사이에 분산시킬 수 있다. 자원을 공유하고자 하는 바램과 병행하여 무단 사용으로부터 자원을 보호하고지 하는 바램도 커지므로, 자원 공유는 대개 자원 보호를 의미한다. 자원 공유 및 자원 보호가 분산형 네트워크 컴퓨터에 있어서 핵심 주제인 점은 놀라운 일이 아니다. 컴퓨터가 더 많이 흔해지고 이들이 서로 접속되면 될수록, 이들 접속된 시스템에서 자원을 보호하는 수단에 대한 필요는 더욱 증대된다. 이러한 필요는 특히 인터넷과 같은 개방형 네트워크의 경우에 특히 절실한 것이다. 이와 같은 개방형의 분산된 네트워크 시스템에 있어서, 무단 접근으로부터 자원, 예를 들어 패스 워드 파일과 개인적인 자료 또는 파일 시스템 공간 및 통신 용량과 같은 하드웨어 자원 또는 암호 서비스와 같은 서비스를 보호하는 것은 필수 불가결하다. 여기서 보호라는 것은 자원 그 자체에 대한 기본적인 접근과 접근한 개체가 그것을 사용하는 것 모두를 의미하는데, 보호란 즉, (예를 들어 개체가 파일 시스템 서비스를 사용할 수 있는) 조대 결정형(coarse-grained) 자원 보호와 (예를 들어 개체가 파일 시스템 서비스를 이용할 수는 있지만 패스 워드 파일에는 접근할 수 없는)미세 결정형(fine-grained) 자원 보호 모두를 의미한다.

분산 시스템에서 자원을 보호하기 위한 현재의 접근 방안은 대부분의 경우에 서비스 제공 개체에서 다소 많은 변화를 필요로 하는 인증 기반 기법에 의존한다. 예를 들어, 1988년 2월에 유제닉스(Usenix)에서 있었던 유제닉스 학술대회 책자의 스타이너(J.G.Steiner), 노이만(B.C.Neuman), 쉴러 (J.I.Schiller)가 'Kerberos: An authentication service for open network systems'라는 제목으로 기술한 바와 같이, 케르베로스(Kerberos)는 각 서비스 요구와 함께 전송될 암호화된 티켓을 필요로 한다. 사용자는 특별하며 신뢰성 있는 티켓 허여 서비스를 통해 인증을 받은 후에 자신이 사용하도록 지정되어 있고 호출하고자 하는 임의의 다른 서비스에 대해 케르베로스 티켓을 요구할 수 있다. 이것은 안전하기는 하지만 자원 보호라는 소기의 목적을 달성하는 데 있어서 번거로우며, 시간 소모적이며 트래픽이 증가하는 방식이다. 더 상세하게는, 자원 보호를 위한 케르베로스 접근 방안은 서비스 제공 개체에 대해 (가시성의 측면에서) 투명하지 않으며, 호출된 개체 자체가 호출 개체로부터의 유효한 티켓에 대해서도 가시적으로 검사해야 한다는 점에서 자유재량형 메커니즘이라는 단점을 지닌다. 개체를 구현하는 코드는 각 라이브러리에 대한 호출을 구비해야 하며, 유효한 티켓 여부에 대한 검사를 잊은 개체는 아무 호출자에게나 그 자원에 대한 접근을 허용할 것이다. 또한 일단 공격자가 중앙 티켓 허가 서비스를 노출시켜 버리면, 그 공격자는 또한 관련된 분산 시스템에 참가한 모든 개체에 대해 접근할 수 있다.

클립톤(D. B. Clifton)은 1995년 11월에 특허된 'Resources access security system for controlling access to resources of a data processing system'이라는 명칭의 미국 특허 제 5,469,556 호에서 자원 보호 과제에 대한 또다른 해법을 제시하였다. 클립톤은 하드웨어 내에서의 메모리 어드레스와 같은 자원을 어드레싱하기 위한, 소위 디스크립터(descriptor)라는 특별한 객체를 소개한다. 테이블 세트가 각 디스크립터에 부착되어 특정 자원에 대한 접근을 제어한다. 본질적으로, 이 디스크립터는 자원의 가상 주소를 형성하여 시스템 내의 자원에 대한 미세 결정형 접근 제어를 구현한다(클립톤 특허 column 3, 34-37 줄의 내용을 참조할 것).

자원 보호를 위한 클립톤의 접근 방안이 케르베로스와 같은 중앙 취약성을 보이지는 않지만, 이 내성이 그 단점의 하나가 되기도 한다. 클립톤의 접근 방안은 디스크립터를 사용함에도 불구하고 메모리 어드레스 번역 메커니즘, 즉 단일 어드레스 공간에 속박되어 있는 분산 시스템에는 적용할 수 없다. 그러나, 분산 시스템은 실제로 정의에 의하면 일종의 통신 매체로 접속된 몇몇 어드레스 공간을 포함한다. 또한, 클립톤의 해법은 투명하지 않고, 사용자가 특별한 디스크립터를 사용해야 하므로 사용자의 프로그램을 부호화해야 한다.

또한, 케르베로스 및 클립톤 접근 방안은 모두 시스템의 실행시간 중(그리고 자원 보호 메커니즘이 설정된 후)에 동적으로 나타나는 자원은 취급할 수 없다. 개체가 외부로부터 분산 시스템 안으로 들어올 수 있는 환경(예를 들어 인터넷으로부터 들어오는 자바 애플릿)에서, 동적으로 생성된 자원이 존재하며 이들 자원에 대해서도 보호할 필요가 있다.

따라서, 본 발명의 주된 목적은 동적 분산 시스템에 적용 가능한 자원 보호 메커니즘을 구현하는 것이다. 본 발명의 또다른 목적은 프로그램 및/또는 보호될 자원에 접근하기를 원하는 개체에 대해 투명한 자원 보호 메커니즘을 생성하는 것이다. 본 발명의 또하나의 목적은 바람직하게는 실행시간 중에 신규로 생성된 자원을 보호하기 위한 동적으로 확장 가능한 자원 보호 방법을 제공하는 것이다.

발명의 요약

분산 시스템 내에서 무단 접근으로부터 보호될 자원은 보통 예를 들어 서버 또는 다른 처리 스테이션과 같은 개체에 의해 소유되거나 또는 제공된다. 원칙상, 개방형의 분산 시스템에서 모든 개체가 다른 개체의 임의의 자원에 접근할 수 있도록 허가되지만, 전술한 바와 같이, 소정의 자원을 보호하고 소정의 개체에 대해서만 및/또는 소정의 조건하에서만 그것에 대한 접근을 허가하는 보안 대책이 요구된다.

자원에는 디스크 공간, 파일, 네트워크 접속 등이 있지만, 또한 객체 및 객체 그 자체에 의해 제공된 방법도 포함할 수 있다. 분산 시스템에서 이와 같은 자원에 대한 접근은 개체 또는 객체가 제공하는 서비스 또는 방법을 통하여 일어난다. 본 발명의 개념은 자원이나 서비스에 대한 접근을 줄이기 위한 아이디어로부터 출발한다. 서비스를 호출하거나 사용하기 위해, 서비스를 소유 또는 제공하는 개체의 이름과 서비스 그 자체의 이름을 모두 알고 있어야 한다. 실제로 서비스에 접근하기 위해, 그 이름을 그 위치에 대응시키는 것이 이루어진다. 즉, 이름이 해결된다. 개체가 알고 있는 이름 전체는 그 개체의 이름 공간을 형성하며, 이와 유사하게 시스템의 개체 위치 전체는 그 시스템의 개체 위치 공간을 구성한다.

따라서 본 발명의 기저에 깔린 개념은, 예를 들어 개체에 대한 서비스의 가시성을 제한하는 것과 같이 이름 해결 과정을 제어함으로써, 서비스에 대한 개체의 접근을 제어하는 것으로 구성된다. 본 발명의 또다른 특징은, 예를 들어 지능형 가로채기(intercept) 관리자에 의해 이 제어를 수행함으로써, 이름 공간 및 이름 해결 과정의 가변 결정립 제어(variable-grained control)를 제공하는 것으로 구성된다. 이와 다른 본 발명의 주 특징은 탈부착 가능한 가드(guard)를 제공하는 데 있는데, 이 가드는 이름과 결합되며 이 이름이 사용되기 전 또는 후에 호출되며 미세 결정립 자원 보호를 위해 제공된다.

이상에서 언급한 하나 이상의 특징을 이용하여 동적으로 구성 가능한 보호 도메인이 생성되는데, 가드 도메인은 제어되고 한정된 환경, 즉 동적으로 구성 가능한 개체가 자유롭게 작동할 수 있는 샌드 박스(sandbox)를 의미한다. 임의의 부당한 행동의 효과는 샌드 박스로 제한되며 따라서 시스템의 나머지 부분으로 전파될 수 없다. 축소 지향적으로 제어된 엔트리 및 이와 같은 샌드 박스로부터/에게로의 탈출점, 특히 미세 조정된 그 모양 및 크기를 또한 동적으로 제공할 수 있다.

도면의 간단한 설명

도 1은 인터페이스 및 자원(용어는 이하에서 정의됨)을 구비하는 개체를 도시한 도면,

도 2a 및 2b는 이름 및 그 객체 표준을 매핑시키는 방법을 도시한 도면,

도 3a 내지 3c는 본 발명에 따른 방법의 단계 및 수정을 도시한 도면,

도 4는 본 발명의 구현을 나타내는 블록도.

발명의 상세한 설명

이하에서, 본 발명의 특징의 일반적인 개관(A)에 대해 먼저 설명하고 이어서 본 발명의 구현(B)을 설명하는데, 이 둘은 모두 객체 재향의 분산 네트워크 시스템에 대한 예로서 자바 기반 네크워크 환경 내에서 다루어진다. 자바란 미국 CA 94043-1100, Mountain View, Garcia Ave., 2550에 소재한 선 마이크로시스템즈 사에 의해 개발된 프로그래밍 언어 및 환경이며, 자바라는 용어는 이 회사의 상표명이다.

A. 원리의 개관

자바 환경

이하에서, 자바 컴퓨팅 환경에 관련되고 본 명세서에서 사용하는 몇 가지의 전형적인 용어를 정리하고 간략히 설명하고자 한다.

'애플릿'은 전형적으로 원격 소스로부터 로딩된 자바 바이트 코드로서, 보통 신뢰할 수 있는 것으로 고려되지 않는다.

'애플리케이션'은 전형적으로 국지(local)로부터 로딩된 자바 바이트 코드로서, 보통 신뢰할 수 있는 것으로서 고려된다.

'인터넷'은 다수의 상이한 서브 네트워크로 구성되는 범세계적인 개방형 컴퓨터 네트워크이다.

'월드 와이드 웹(World Wide Web: WWW)'은 인터넷의 개방형 서브 네트워크이다.

'인트라넷'은 폐쇄된 내부 네트워크로서, 보통 인터넷의 서브 네트워크로서 회사 내에서 종종 사용된다.

'브라우저'는 월드 와이드 웹 내에서의 둘러보기(서핑)를 지원하고 애플릿의 다운로딩 및 실행을 위한 컴퓨터 소프트웨어 또는 환경이다.

'자바 바이트 코드'는 이하에서 언급할 JVM에 대한 컴파일러에 의해 생성된 코드이다.

자바 가상 머신(Java Virtual Machine: JVM)은 가상적이지만 자바 바이트 코드를 실행하는 잘 규정된 프로세서이다. 머신이 소프트웨어로 구현되기 때문에 가상적이라고 한다.

'라이브러리'는 프로그램에 사용되는 정의된 함수 또는 클래스의 집합이다.

'샌드 박스'는 자바 애플릿이 수용되어 있는 운동장이다. 그것은 정의된 입구 및 출구 점(문)을 구비한 경계에 의해 묘사된 실행 환경이다(이에 대한 상세한 설명은 아래 내용을 참조할 것).

샌드 박스 보안 모델

앞서 언급한 선 마이크로시스템즈 사의 프리징거(J.S.Fritzinger)와 뮬러(M.Mueller)에 의해 기술된 소위 샌드 박스는 자바 애플릿에 대해 정의된 경계 및 게이트에 의해 묘사된 운동장을 제공한다. 모든 애플릿은 이 샌드 박스에 수용된다. 샌드 박스 외부에 있는 어떤 것에 접근하기 위해, 애플릿은 해당 게이트를 통과해야만 한다. 이와 유사하게, 어떤 애플릿과의 통신도 통과를 통제하고 제한할 수 있는 게이트를 통과해야만 한다. 전형적으로, 애플릿은 원격으로 개발된다. 사용자는 습관적으로 애플릿의 다운로드를 시

작하며, 그 애플릿은 보통 인터넷과 같은 신뢰성 없는 공공 네트워크 상을 돌아다닌다.

애플릿을 신뢰할 수 있는가 아닌가는 다음 요소:

애플릿 프로그래머가 적대적 프로그램을 작성할 수 없도록 되어 있는가, 즉 프로그래머를 신뢰할 수 있는가와,

언어 사양에 정의된 규칙에 따라 바이트 코드를 생성하는 컴파일러를 신뢰할 수 있는가와,

침입자가 애플릿의 통합성을 손상시키거나 수정할 수 있다는 점에서, 네트워크를 신뢰할 수 있는가에 본질적으로 의존한다.

이상에서 언급한 사항 중 하나라도 부정적인 평가를 받는 애플릿은 신뢰성이 없다고 고려된다.

반면에 독립형 애플리케이션은 대개 국지적으로, 즉 안전한 환경내에서 개발되거나 또는 인트라넷과 같은 신뢰성 있는 네트워크 상을 돌아다닌다. 따라서 애플리케이션은 보통 신뢰할 만한 것으로 고려된다. 그러나 애플릿과 애플리케이션을 흑백 논리로 구별하는 것만으로는 충분하지 않다. 샌드 박스식 접근 방안은 애플릿에만 적용되고 애플리케이션에는 적용되지 않기 때문에 구현된 보안은 애플릿에 한정된다. 또한 현재 구현된 샌드 박스는 불충분하고 불완전하기 때문에 결함이 다수 발견되고 있다.

보통 샌드 박스는 브라우저 내에서 구현된다. 그것은 최종 사용자와 교감이 없이 하드 코딩되므로 프로그램 가능하지 않다. 따라서 그것은 고정된 경계 및 고정된 게이트를 갖는 정적 모델이다. 게이트 그 자체, 즉 그 위치, 그 정책 또는 그 존재는 바뀔 수 없고 오직 게이트의 결정만을 바꿀 수 있다. 이 모델은 다운 로딩된 코드의 기원은 고려하지 않으며, 오직 국지냐 원격 프로그램이냐 즉 애플리케이션인가 애플릿인가를 개략적으로 구분할 뿐이다.

요컨대, 이미 구현된 자바 보안 모델은 적절한 보안 정책의 실시와 그 정책을 구현하는 메커니즘을 분리하지 못하였다. 대부분의 사용자가 제어할 수 있는 결정이 자바 가상 머신 내에서 하드 코딩되어 있으므로 알려진 보안 모델은 정적이고, 샌드 박스 패러다임의 구현은 충분하지 못하다. 확실한 것은, 이 알려진 모델이 보안 특징을 구성할 수 없음으로 인해 이를 수용하고 적용하는 데는 한계가 있다는 것이다.

본 발명에 따른 객체 지향 시스템에 대한 새로운 보안 메커니즘은 자바 환경에 직접 적용될 수 있다. 이 장에서 필수적인 일부 용어는 잘 받아들여지도록 정의된 반면, 어떤 것은 이 설명에서 특정한 의미로 사용된다. 용어 사용의 일관성을 위해, 이하에서 이들 용어에 대해 간략히 정의하고자 한다.

용어 정의

'객체'는 데이터를 함유하고 방법을 제공하는데, 데이터는 이 방법에 의해서만 접근될 수 있다. 객체는 동적으로 생성된다. 즉 그것은 즉시적이다.

'클래스'는 객체에 대한 정의이며 정적이다.

'개체'는 그것이 능동형인지 또는 수동형인지에 관계없이 객체 또는 참조 클래스로서 고려된다. 개체는 자원을 소유 또는 제공한다.

'인터페이스'는 개체 데이터 및 서비스가 접근되고 변환되는 개체의 방법의 세트로 구성된다.

'이름'은 개체의 방법에 대한 상징 참조인데, 이름은 실행시간 시스템의 실행 요소에 의해서는 이해, 즉 해석되지 않는다.

'객체 참조'는 개체의 방법의 위치에 대한 지시자인데, 객체 참조는 실행시간 시스템의 실행 환경에 의해 이해, 즉 해석된다.

'서비스'는 객체 참조와 상징 참조 중의 하나를 통해 지시된 개체의 방법으로서 고려된다.

본 발명

독창적인 보안 메커니즘의 목적은 객체 지향 시스템에서 무단 접근으로부터 자원을 보호하는 것이다. 이것은 접근 제어뿐만 아니라 자원 활용의 감사(audit) 및 감시(monitor) 용도로 사용될 수 있다. 이것의 설계는 자원에 대한 접근이 엄격히 제어되면 어떤 프로그램도 해를 끼치지 않는다는 기본적인 가정에 기초한다. 다음에 기술할 요소들은 단독으로 또는 서로 조합하여 본 발명에 따른 보안 메커니즘에 필수적인 것이다.

객체 지향 패러다임: 데이터가 개체에 함유되고, 제공된 인터페이스에 의해서만 수정될 수 있다. 개체 데이터를 보호하기 위한 접근 방안은 이 인터페이스에 대한 접근을 통제하는 것이다.

태만 객체 결합(lazy object binding): 각 상징 참조(또는 이름)는 실행시간에서, 그것을 통해 서비스가 호출될 수 있는 해당 객체 참조로 대체된다. 이 대체 과정은 본 발명에 따른 보안 메커니즘에 의해 가로채기(intercept) 된다.

이름 해결(name resolution): 주어진 이름(상징 참조)에 대한 객체 참조를 얻기 위해, 이름 해결이 필요하다. 이 이름 해결에서는 이름을 객체 참조에 대응시킨다. 본 발명의 독창적인 메커니즘에서, 이 대응은 이름 해결을 제어하는 가로채기 관리자에 의해 가로채기될 수 있다.

가드 객체(guard object): 이것은 객체 참조에 임의로 부착될 수 있다. 이 가드는 객체 참조가 사용되기 전 및/또는 사용된 후에 호출된다. 또한 이것은 콘텍스트 정보에 접근할 수도 있다.

객체 지향 패러다임

자원은 개체에 함유되고 제공된 인터페이스를 통해서만 접근될 수 있다. 따라서, 객체 지향 시스템에서 무단 접근으로부터 자원을 보호하기 위해, 그런 자원에 대한 접근 통제 이슈를 관련된 개체의 인터페이스에 대한 접근 통제의 문제로 한정할 수 있다. 개체는 상이한 자원을 제공할 수도 있고 따라서 도 1에 도

시한 바와 같이 상이한 인터페이스를 제공할 수도 있다. 특정 자원을 사용하기 위해, 이 자원의 이름(그리고 위치)을 알아야 하며, 이 이름은 그 다음에 해당 방법이 존재하는 위치를 가리키는 객체 참조로 변환된다.

태만 객체 결합

결합은 실행 가능한 코드의 필요한 요소를 함께 연결하는 과정이다. 요소는 컴파일러가 생성하며 객체 파일로 불린다. 객체 파일은 다른 객체 파일에 대한 상징 참조를 구비한다. 연결자(linker)는 이 객체 파일들을 함께 연결함으로써 상징 참조를 해당 객체 참조로 대체한다. 이 시점에서, 실행 가능한 코드는 실행 환경에 의해 이해 가능, 즉 해석 가능한 필요한 객체 참조를 보유하며 이에 따라 실행 가능한 코드는 참조 방법을 호출할 수 있다.

대체가 실행시간에 수행되면, 그것은 '동적 결합'으로 지칭되며, 그렇지 않으면, 즉, 대체가 코드가 실행되기 전에 수행되면, 그것은 '정적 결합'으로 지칭된다.

모든 상징 참조의 대체가 한 시점에 일어나면 연결 과정은 '열성 결합(eager binding)'으로 불린다. 상징 이름에 대한 객체 참조가 실행시간 중에 필요한 것이 오직 한 번 수행된 경우에는, 그것은 '태만' 결합이라 불린다. 대체가 일단 실행 가능한 코드로 수행되면, 그것은 고정되고 더 이상 변화되지 않는다.

예를 들어 C++ 컴파일러는 사람이 읽을 수 있는, 다른 개체(즉, 이름)의 상징으로 참조된 방법을 그 유형 정보와 병합하여 추가 상징 정보를 형성한다. 이들 참조 중의 어떤 것도 여전히 상징(즉 이름)이지만, 더 이상 사람이 읽을 수 있는 포맷은 아니다. 결합자(binder)는 상징 참조에 따라 적절한 코드를 포함하고 상징 참조를 적절한 객체 참조로 대체한다. 객체 참조는 방법이 존재하는 위치의 시작을 가리킨다. 자바 환경은 태만 결합 접근 방안을 사용한다. 각 상징 참조는 일단 실행 환경이 그것을 사용하면 대체된다.

분산 접근 방안에서는 어드레스 공간 및/또는 시스템을 통한 방법의 호출이 가능하다. 대부분의 경우에, 원격 객체(remote object)는 국지 대표(local representative)로서 행동하는 대리 객체에 의해 표현된다. 보통 대리 객체(proxy object)는 원격 객체를 찾고, 생성하고, 파괴하며 호출한다. 따라서, 객체 참조는 해당 대리 객체의 위치를 가리킨다.

가로채기 및 이름 해결

본 발명의 독창적인 메커니즘의 특징은 결합 과정이 열성 결합인지 또는 태만 결합인지에 상관없이 그것을 가로채기하고, 가로채기 관리자를 삽입한다. 상징 참조, 즉 이름의 해결은 후자와 해당 객체 참조 사이의 대응을 필요로 한다. 주어진 상징 참조(이름)에 대한 해당 객체 참조를 발견하는 과정이 이름 해결 과정이다. 이것은 도 2a 및 2b에 도시한 바와 같은 이름 해결 과정에 의해 실행된다. 특정 방법을 이용하기 위해, 호출하는 개체는 적절히 객체 참조로 해결되어야 하는 상징 참조 또는 이름을 알아야만 한다. 개체가 알지도 모르며 그 상징 참조로 표현되는 참조와 개체가 확실히 알고 있으며 그 객체 참조에 의해 표현되는 참조들을 분리하면 다음의 두 개의 상이한 이름 공간이 생긴다. 첫째로, 개체의 '추정 이름 공간(concrete name space)'은 상징 참조의 세트로 구성된다. 이들 상징 이름은 아직 해결되지 않았지만, 해결 과정의 대상이다. 개체는 개체는 이들 참조가 존재한다고 추정하지만, 이 추정은 아직 해결 과정을 통해 검증되지 않았다. 이들 방법은 아직 호출될 수 없으며, 실행시간 실행 요소에 의해 이해될 수 없다. 둘째로, 개체의 '확정 이름 공간'은 객체 참조의 세트로 구성된다. 이 객체 참조는 해결된 이름이고 그것은 해결 과정을 통해 검증되었기 때문에 개체는 명백히 알고 있다. 객체 참조는 실행시간 실행 환경이 이해할 수 있다.

'확정 이름 공간'에서 지적된 방법 세트는 보통 추정 이름 공간에서 지적된 방법의 부분 집합이다. 참조들은 다르지만(객체 참조와 상징 참조), 그것에 의해 지적된 방법은 같다.

가로채기 관리자

도 2b에 도시한 소위 가로채기 관리자는 선택하기에 따라 제공될 수 있는 본 발명의 한 특징이다. 가로채기 관리자가 제공될 경우 그 주된 임무는 특정 보안 정책에 따라 이름 해결 과정을 제어하고 수정하는 것이다. 예를 들어, 그것은 몇 가지 상징 이름의 해결을 막음으로써 확정 이름 공간을 축소시킬 수 있다. 이름 해결 과정을 가로채기하고 가로채기 관리자를 삽입하면 접근 제어 메커니즘으로 간다. 보안 목적으로 확정 이름 공간을 규제하는 것은 어떤 자원도 그것을 대표하는 개체의 객체 참조를 구비하지 않고 사용될 수 없다는 가정에 기초한다. 가로채기 관리자가 이 해결을 허가하는 경우에만 상징 참조된 방법이 해결된다.

도 2b에 도시한 바와 같이, 가로채기 관리자는 이름 해결 과정의 일부이다. 이름 해결 과정의 결과인 객체 참조가 가로채기되는데, 가로채기 관리자는 수정되지 않은 객체 참조나 가드 객체가 부착된 객체 참조 또는 예외적으로 참조된 개체가 존재하지 않는다는 통지를 반송한다. 가로채기 관리자에 의해 수행된 접근 제어의 조밀도는 해결 과정의 조밀도에 의존한다. 더 정확히 설명하면, 가로채기 관리자는 미세 레벨이 아닌 조대 레벨 상에서 행동할 수 있다. 그럼에도 불구하고 가변 결정립 접근 제어는 여전히 존재한다. 가로채기 관리자는 소스 개체 이름, 목적지 개체 이름 및 그들의 데이터와 같은 콘텍스트 민감성 정보에 접근한다. 접근 하나만 제어하는 것은 어떤 경우에는 충분하지 않으며, 자원의 사용을 감시하거나 감사하는 것이 필요할 수 있다. 이것은 보호된 자원이 사용되기 전 및/또는 후에 보안 메커니즘의 요소를 더 호출할 것을 요구한다. 이와 같은 객체는 가드라고 불리며, 가로채기 관리자에 의해 삽입될 수 있다.

가드 객체

본 발명의 또다른 선택에 따른 특징인 가드 객체는 특정 객체 참조와 결합하며, 가로채기 관리자가 어떻게 가드 객체를 설치했는가에 따라 객체 참조가 사용되기 전 및/또는 후에 호출될 것이다.

관련된 객체 참조가 사용되기 전에 가드 객체가 호출되어 목적지 개체를 위해 공급된 아규먼트(argument)에 대해 접근하면 가드 객체는 목적지 개체 전에 설치되었다고 일컬어진다. 관련된 객체 참조가 사용된 후에 가드 객체가 호출되어, 호출된 메시지가 반송되어 그 반송 값에 접근하면 가드 객체는 목적지 개체

이후에 설치되었다고 일컬어진다. 도 2b는 가드 객체가 부착된 수정된 객체 참조를 가로채기된 이름 해결 과정의 반송 결과의 하나로서 도시한다.

가드 객체에 에러가 없으면, 실행이 정상적으로 계속된다. 가로채기 관리자가 사용한 콘텍스트 정보와 유사하게, 가드 객체는 또한 콘텍스트 민감성 정보에 대해 접근한다. 여기에는 소스 개체 이름, 목적지 개체 이름, 호출 파라미터가 포함된다. 가드 객체가 목적지 개체 전에 설치되면, 그것은 다음과 같은 행동:

-호출을 거절하는 것,

-호출을 변경하지 않고 통과시키는 것,

-호출을 공급된 데이터로 변환시키는 것,

-호출의 목적지를 바꾸는 것,

-호출자 개체 대한 권리를 할당 및/또는 검사하는 것,

-접근을 감시하는 것,

-통지 및/또는 회계 감사 서비스를 수행한다.

가드 객체가 목적지 개체 이후에 설치되면, 그것은 다음과 같은 행동:

-반송 파라미터 또는 상태를 변경하는 것,

-사전 할당된 권리를 제거하는 것,

-응답을 감시하는 것,

-통지 및/또는 회계 감사 서비스를 수행한다.

가드 객체는 소스 또는 목적지, 즉 호출한 개체 및 호출된 개체에 의해 영향받지 않는데, 이는 그것이 가로채기 관리자에 의해 외부로부터 부착되었기 때문이다.

가드 객체는, 호출한 또는 호출된 개체에 관한 한, 불 수 없고 투명하다. 가로채기 관리자와는 반대로 가드 객체는 확정 이름 공간을 직접 축소할 수 없으며, 간접적으로만 영향을 미칠 수 있다. 하지만, 그것이 부착된 객체 참조를 역해결하여 원래의 상징 참조로 만들 수 있다. 이와 같은 역해결의 결과로서, 이 방법을 그 다음에 사용할 때에는 상징 참조를 다시 만나고 이름 해결 및 가로채기 관리자를 위한 호출로 이어질 것이다. 가로채기 관리자는 다시 그 보안 정책에 따라 호출 개체의 확정 이름 공간을 수정할 가능성을 갖는다. 객체 참조의 역해결은 상징 참조가 해결된 다음에 보안 정책이 변하는 경우에 유용할 것이다.

또한, 각각의 가드 객체는 그것이 얼마나 오랫동안 객체 참조에 부착되어 있기를 원하는지에 대해 결정할 가능성이 있다. 가드 객체는 스스로 분리되도록 결정할 수도 있으며, 따라서 이 경우에는 객체 참조의 부름에 더 이상 호출되지 않을 것이다.

능력

능력이란 특정 모드에서 특정 객체를 호출할 권리이다(이에 대해서는 이후에 보다 상세히 설명할 것임). 1995년에 Prentice Hall International 사가 간행한 'Modern Operating System'이라는 명칭의 자료에서 타넨바움(A.S. Tanenbaum)이 기술한 바와 같이, 권리는 다음의 세 요소를 포함할 수 있다.

(1) 객체에 대한 지시자(pointer): 필요한 지시자는 해결된 참조(객체 참조)에 의해 성취된다. 실행 가능한 코드가 지시자를 구비하면, 적절한 실행시간 시스템이 호출을 가능하게 한다고 가정한다. 이것은 능력을 구비하는 모든 개체가 해당 방법을 호출할 수 있음을 의미한다.

(2) 객체의 유형 정보: 요구된 유형 정보는 부착된 가드 객체를 통해 성취된다. 능력의 유형은 관련된 가드 객체에 의존한다.

(3) 객체에 대한 접근 권리: 요구된 접근 권리는 부착된 가드 객체에 의해 성취된다. 그것은 권리를 검사할 수 있고, 호출한 개체는 특정 방법을 사용하기 위해 권리를 제공해야 한다. 또한, 호출한 개체가 적절한 권리를 구비하면, 가드 객체는 이 서비스를 이용하기 위해 현존하는 권리를 강화하거나 또는 신규 권리를 할당할 수도 있다.

객체 참조와 가드 객체는 능력이 유사하지만 동일하지는 않다. 설치된 가드 객체를 구비한 해결된 이름은 향상된 동적 능력으로 간주될 수도 있다. 능력의 동적 요소는 결합된 가드 객체이다.

개관 요약

객체 지향 시스템에서의 자원 보호를 위해 제안된 접근 방안은 다음 특징 중 하나 이상을 포함한다. 객체 지향 패러다임은 객체 및 그 인터페이스를 통해 자원에 대한 접근을 잘 제어하게 한다. 접근 통제는 호출한 개체에 대한 이름 공간의 제어를 통해 성취되어 해결된 이름의 가시성을 제어한다. 호출한 개체가 서비스를 볼 수 없다면 서비스를 호출할 수 없으며, 방법을 호출할 가능성이 없이 관련된 자원을 사용할 수 없다. 해결 관리자(그리고 선택에 따른 가드 객체)가 콘텍스트 정보를 이용할 수도 있기 때문에, 매우 유연하게 행동할 수 있으며 콘텍스트에 의존적이다. 가드 객체는 개체가 누구인지 모르고도 외부로부터 부착될 수도 있는데, 그것은 호출한 및 호출된 개체에 관한 한 가시적이지 않으며 투명하다. 따라서, 메커니즘은 성능 부담 면에서 적은 비용으로, 그리고 동적 형태로 보호한다.

B. 구현

이하에서 본 발명에 따른 보안 메커니즘의 구현이 자바 환경에서 기술된다. 해석기(interpreter) 및 실행시간 라이브러리를 포함하는 자바 가상 머신(Java Virtual Machine: JVM)에 대해 이루어진 모든 변화에 대

해 살펴볼 것이다. 또한, 자바 내에서 구현된 본 발명의 보안 메커니즘의 요소 또한 제시될 것이다.

이 장은 또한 구현의 상세 내용을 취급하기 때문에, 당업자라면 자바 프로그램 언어 그 자체뿐만 아니라 JVM 및 C 언어에서의 그 구현의 기본 메커니즘에 친숙할 것이라고 가정한다. 자바 언어에 대한 포괄적인 제시 및 JVM에서의 구현은 1996년에 Addison-Wesley 출판사에 의해 간행된 고슬린(J. Goslin) 및 텔레(G. Tele)에 의한 'The Java Language Specification'라는 명칭의 책과 린드홀름(T. Lindholm) 및 옐린(F.Yellin)에 의한 'The Java Virtual Machine Specification'라는 명칭의 책에 기술되어 있다. 본 발명의 보안 메커니즘의 구현 요소들은 자바 개발 키트(Java Development Kit: JDK)) 버전 1.0.2 및 JVM의 AIX 버전에 기반한다. 이 장은 시스템 클래스로 표현된 보호 시스템 자원의 요소에 대해 기술한다. 자바 가상 머신의 변화는, 약간의 변화는 불가피했지만, 가능한 한 최소한으로 유지되었다.

B1. 기능 개요

본 발명의 보안 메커니즘의 구현된 기능은 가로채기 관리자 및 몇가지 기본적인 가드 객체의 구현을 포함한다. 개체라는 용어는 특정 클래스에 적용하기 위해 이전 장에서 상세히 설명되었다. 클래스는 그것이 자원을 소유하고 나타낸다는 면에서 개체라고 볼 수 있다. 시스템 자원은 자바 라이브러리 내에서 그 해당 시스템 클래스를 통해 표현되기 때문에, 클래스는 모호하지 않고 충분히 공인된 이름으로서 알려져 있다. 따라서 이 구현은 자바 라이브러리 내에 존재하는 시스템 클래스를 보호한다. 이름 해결 과정(이하 참조)으로부터 수신된 객체 참조는 실질적으로 방법 서술의 시작을 가리키는 코드의 해당 부분에 대한 지시자이다. 따라서, 객체 참조는 위치에 대한 지식을 포함하고 있어서 방법은 즉시 호출될 수 있다.

시스템 클래스의 보호는 이름 공간을 수정함으로써 그것에 대한 접근 제어를 통해 성취된다. 객체의 확정 이름 공간이 객체 참조를 포함한다면 참조된 방법에 대한 후속 접근이 허가된다. 따라서 위의 가정에서 서술했듯이, 프로그램은 객체 참조를 이용할 수 있는 자원들에만 접근할 수 있다. 객체의 확정 이름 공간을 제어하고 수정하기 위해, 태만 결합 메커니즘 및 이름 해결은 전술한 바와 같이 가로채기 관리자를 삽입함으로써 이용된다. 가드 객체는 객체 참조와 결합된다. 원래의 방법이 실행되기 전에 가드 객체를 호출하기 위해, 방법 호출 기법은 설치된 가드 객체에 대한 검사를 구비하도록 확장된다. 이 기능을 구현하기 위해, 다음과 같은 주제, 즉

- 이름 공간을 통한 자원 보호,

- 자바 가상 머신에 있어서의 방법 호출 요소의 변경,

- 본래의 C 코드에 접근하는 가로채기 정보 기반 클래스,

- 가로채기 관리자의 구현,

- 가드 객체 기반 클래스

가 매우 중요하다.

B2. 자바 가상 머신 변경-시스템 레벨

이 섹션에서는 자바 가상 머신 및 자바 실행시간 라이브러리의 강화를 논의한다. 강화는 본래의 C 코드에서 구현되며 따라서 머신 의존적이다. 본 발명에 따른 보안 메커니즘을 구현하도록 다음 요소, 즉

- 가상 머신의 이름 해결 요소,

- 방법 호출을 취급하는 모든 op 코드(opcode),

- 메모리에서의 이진 방법 표현,

- 실(thread) 구조

가 변경되었다.

이름 공간 수정

접근 통제를 달성하고 그 결과로서 자원 보호를 성취하기 위해, 호출한 객체의 확정 이름 공간이 제어되고, 그것이 축소되거나 확장된다.

시작하는 객체의 이름 공간을 수정하기 위해, 자바의 태만 객체 결합 방법이 이용된다. 자바의 이진 파일 레이아웃은 다른 것의 클래스, 인터페이스 및 그들의 필드, 방법, 제작자를 충분히 공인된 이름을 이용하여 상징 면에서 참조한다. 필드 및 방법에 대해, 이들 상징 참조는 필드 또는 방법 그 자체의 이름뿐만 아니라, 그 필드 또는 방법을 선언하는 클래스 또는 인터페이스 유형의 이름을 적절한 유형 정보와 함께 포함한다. 객체가 사용될 수 있기 전에, 그 이름이 해결되어 객체 참조가 되어야 하는데, 식별자는 올바른지 검증되며, 전형적으로는 참조가 반복적으로 사용되면 더 효율적으로 처리될 수 있는 직접적인 객체 참조로 교체된다. 에러가 생기면, 예외가 발생하는데, 이에 대해서는 1996년에 Addison-Wesley 출판사에 의해 간행된 고슬린(J. Goslin), 조이(B. Joy), 텔레(G. Tele)에 의한 'The Java Language Specification'이라는 명칭의 책 및 린드홀름(T. Lindholm)과 옐린(F.Yellin)에 의한 'The Java Virtual Machine Specification'라는 명칭의 자료를 참조하면 된다.

이름이 객체 참조로 해결된 후에, 가로채기 관리자는 호출되어 상징 참조의 해결이 허용되어야 하는지의 여부를 결정한다(도 4를 참조할 것). 따라서, 이름 해결은 가로채기되고, 따라서 태만 객체 결합이 이루어진다. 가로채기 관리자의 반송 값에 따라, 확정 이름 공간의 수정이 다음에 기술하는 바와 같이 수행된다.

가로채기 관리자가 '거짓'이라는 반송을 하면, NoClassDefFoundError 예외가 제공되어, 클래스 파일이 존재하지 않는다는 것을 호출한 객체에게 표시한다. 따라서, 원하는 자원의 이름은 해결될 수 없고 결과적

으로, 코드의 해당 부분의 위치는 알려지지 않는다. 가로채기 관리자에게 묻고 이어서 예외를 제공하는 이 기법은 호출한 객체의 확정 이름 공간을 축소한다.

가로채기 관리자가 '참'이라는 반송을 하면, 호출이 정상적으로 계속되고 확정 이름 공간은 수정되지 않는다. 사실, 확정 이름 공간은 해결된 이름으로서, 즉 객체 참조를 구비하는 코드의 해당 부분에 대한 지시자로서 확장된다.

이하의 코드 부분 C1은 이름 해결 기능에 삽입된 신규 코드를 도시한다. 이 C 코드 부분은 다음 단계:

(1) 가드 객체를 관리하기 위해 필요한 구조를 초기화하는 것,

(2) 요구한 해결이 허가될 지에 대해 검사하여, 허가되지 않으면 방법을 미해결로 표시하고 '거짓'을 반송하는 것,

(3) 그렇지 않으면, 방법을 해결로 표시하고 상징 정보를 코드의 해당 부분에 대한 지시자로 대체하는 것

을 수행한다.

가로채기 관리자가 몇 개의 가드 객체를 부착할 수 있기 때문에, 방법 호출 기법은 이름 해결 과정 뿐만 아니라 수정을 필요로 하는 것이다.

코드 부분 C1

이 코드 부분은 상징 참조된 이름을 해결하는 기능에 삽입되었다. 적절한 예외가 ic_CheckMethodResolving()method에 제공된다(이 코드에서 가드 객체는 '필터 객체'로 불리운다).

```
[해결 방법]
/*
해당 방법 표현이 발견될 수 있으면, 이 방법에 대한 필터 발송표를 준비함
*/
ic_PrepareFilter(class, mb);
/*
이 특정 이름 해결이 허가되는지의 여부를 검사함
*/
if (!ic_CheckMethodResolving(mb)){
/*
이 방법의 해결이 허가되지 않으면 그것을 미해결로 표시함
*/
fieldfiltertable(&mb->fb)->resolved = FALSE;
return FALSE
}
/*
방법을 해결로 표시함
*/
fieldfiltertable(&mb->fb)->resolved = TRUE
상징 이름 정보를 적절한 방법블럭에 대한 지시자로 변경함
*/
constant_pool[index].p = mb;
continue_as_usual();
[...]
```

방법 호출

자바 해석기는 이미 해결된 방법의 실행을 책임진다. 방법 호출과 관련된 절차에 관한 상세한 논의는 1996년에 Addison-Wesley 출판사에 의해 간행된 린드홈름(T. Lindholm) 및 옐린(F.Yellin)에 의한 'The Java Virtual Machine Specification'이라는 명칭의 책에 기술되어 있다. 도 4는 방법 호출의 제어 흐름을 간략히 정리하여 도시한다. 도 4에서 본 발명에 의해 첨가된 10 개의 신규 요소는 19-25 및 27-29이며, 다른 요소는 원래의 자바 요소이다. 본 발명에 따른 신규 요소는 이름 해결에 관한 검사 및 가드 객체의 실행을 포함한다.

op코드

op코드는 특정 세트의 동작을 나타내는 짧은 용어로서 보통 숫자이다. 그것은 특정 기능을 나타낸다. 예를 들어, 연상 기억 코드 opc_invokevirtual과 관련되는 op 코드 182 (0xb6)는 가상 방법을 호출하기 위

한 기본 단계를 나타낸다. 다양한 자바 코드 및 그것이 제공하는 기능에 대한 상세한 설명은 앞서 언급한 린드홀름(T. Lindholm)의 책에서 찾을 수 있다. 가드 객체를 설치하고 유지하기 위해, 방법 호출을 취급하는 op 코드의 구현이 변경된다. 가드 객체에 대한 검사와 그것의 후속 실행을 위한 코드를 구비하도록 기능이 확장된다. 마지막 가드 객체가 실행된 후에, 원래의 방법이 실행된다. C2는 방법 호출을 취급하는 모든 op 코드에 삽입된 C 코드 부분이다. 이 코드 부분은 다음에 기술할 단계:

(1) 가드 객체가 설치되었는지의 여부를 검사하여, 설치된 가드 객체가 없으면 정상적으로 계속하는 단계와,

(2) 방법이 해결되었는지의 여부를 검사하여, 그것이 해결되었으면 이어서 설치된 가드 객체를 호출하는 단계와,

(3) 예외가 취급되었는지의 여부를 검사하여, 예외가 제공되었으면 방법의 실행을 중단하고 그것을 취급하는 단계를 수행한다.

가드 객체를 객체 참조에 부착하기 위해, 원래 방법의 메모리 레이아웃이 확장된다.

코드 부분 C2

이 코드 부분은 원래 방법의 실행 전에 삽입된다.

[실행 방법]

```
/*
필터 발송표가 설치되었나?
설치되었다면 필터 객체가 설치되었는지의 여부를 검사함
*/
if (fieldfiltertable(&mb->fb)&&
  (fieldfiltertable(&mb->fb)->used_filter)){
/*
방법이 본 발명의 메커니즘을 통해 해결되었는가?
*/
if (fieldfiltertable(&mb->fb)->resolved == TRUE) {
/*
* 설치된 필터객체를 호출함
*/
ic_InvokeFilterObjects(fieldfiltertable(&mb->), ee);
} else {
/*
이름 공간을 검사하여 가로채기 관리자 객체를 호출함
*/
ic_CheckMethodResolving(mb);
}
/*
모든 것이 잘 되었는가? 아무도 예외를 제공하지 않았음
*/
if (exceptionOccurred(ee))
  goto handle_exception;
}
continue_executing_original_method();
[...]
```

방법 표현

객체의 임의의 방법을 실행하기 위해, 방법은 메모리 내에 해당 이진 표현을 필요로 한다. 방법의 이진 표현은 정적 요소와 동적 요소로 나뉜다. 정적 요소는 그것의 클래스 묘사와 정적 필드를 포함하고, 동적 요소는 그것의 동적 필드를 포함한다. UNIX 운영 체제 내에서 실행 가능한 이진 표현의 텍스트 세그먼트와 데이터 세그먼트는 유사한 개념이며, 두 세그먼트는 어드레스 공간의 상이한 영역으로 대응되는데, 텍스트 세그먼트는 읽기 전용 모드로 대응되고 데이터 세그먼트는 읽기/쓰기 모드로 대응된다.

빙법의 정적 요소, 동일한 클래스 묘사로부터 실증된 모든 객체에 의해 공유되지만, 각 객체는 그 자신만의 동적 요소를 갖는다. 가드 객체는 방법의 정적 요소(fieldblock structure)에 부착된다. 따라서, 가드 객체는 또한 같은 클래스 서술의 모든 경우에 의해 또한 공유된다. 가드 객체가 정적 요소에 부착되기는 하지만, 가드 객체를 클래스 및 그 방법에 부착시키기보다는 상이한 객체 및 그 방법에 부착시키도록 구현된다.

가드 발송표

가드 발송표는 부착된 가드 객체를 관리한다. 이는 각 가드 객체의 참조를 지니며 얼마나 많은 가드 객체가 설치되는지를 표시한다. 헤더는 다음의 관리 요소:

얼마나 많은 가드 객체가 사용되는지를 표시하는 변수(used_filter로 표기됨)와,

공간을 할당받은 가드 객체의 수를 표시하는 변수(res_filter로 표기되는데, res_filter는 used_filter보다 크다는 것이 언제나 성립해야 함)와,

방법이 해결된 것으로 고려되는지 아닌지를 표시하는 플래그(resolved로 표기됨),

부착된 방법 표현에 관한 지시자(mb로 표기됨)와,

호출 객체의 클래스 파일에 대한 지시자(caller로 표기됨)와,

관리되는 가드 객체 구조를 포함하는 동적 어레이의 제 1 요소에 대한 지시자

로 구성된다. (역사적인 이유로, 가드 객체는 코드 내에서 필터 객체로 불리운다)

다음 코드 부분 C3은 가드 발송표 헤더의 구조를 도시한다.

코드 부분 C3

이것은 가드 발송표의 헤더이다.

```
struct filterdtable {
  int used_filter;
  int res_filter;
  int resolved;
  struct methodblock *mb;
  struct ClassClass *caller;
  struct filterdesc *filters[1];
};
```

filters로 표기되는 어레이 요소의 유형은 코드 부분 C4에 도시되는데, 그것은 다음 요소:

가드 객체의 객체 메모리 위치에 대한 지시자(obj로 표기됨)와,

호출되어야 할 방법의 이름(methodname으로 표기됨)과,

두 방법이 동일한 이름을 갖지만 상이한 서명을 가지는 것을 구현할 수 있음으로 인한 그 해당 서명(signature로 표기됨),

가드 객체가 그 자신을 해당 객체로부터 분리시킬 때 호출된 가로채기 관리자 클래스에서의 방법을 지정하는 응답 호출 방법 명(notify로 표기됨)

을 포함한다.

코드 부분 C4

필터 발송표에서의 각 요소는 filterdesc의 유형이다.

```
struct filterdesc {
  HObject *obj;
  char *signature;
  char *methodname;
  char *notify;
};
```

가드 객체는 자바로 구현되기 때문에, 이는 이름 공간 수정 및 부착하는 가드 객체에 대한 잠재적인 후보이다. 따라서, 반복 문제를 피하기 위해 그에 따라 실행 실(executing thread) 표시자가 붙는다.

실

실은 어드레스 공간 내에서의 실행 경로이다. 이 어드레스 공간은 현재 실행되고 있는 많은 방법에 의해 공유될 수 있다. 실 개념에 관한 상세한 논의는 1989년 1월 6일 발행된 디지털 장비사(Digital Equipment Corp.)의 System Research Center (SRC) Report에 게재된 비렐(A.D. Birell)에 의한 'An Introduction to Programming with Threads' 라는 명칭의 자료에 나타나 있다.

본 발명에 따른 가드 객체 및 가로채기 관리자는 자바로 구현된다. 이들은또한 자원에 접근하기 때문에, 본 발명에 따른 메커니즘에 속하는 요소와 그렇지 않은 요소를 구별하기 위한 기준이 필요하다. 그렇지 않으면, 가로채기 관리자가 자원에 접근하고 이어서 방법을 호출할 때마다, 그것은 가로채기 관리자 자신

이 검사한다. 이 반복은 실을 적절히 표시하면 없어질 수 있다. 실이 가드 객체 또는 가로채기 관리자에 들어가면, 실은 감독 상태에 있는 것으로 표시된다. 다음에 실이 가드 객체 또는 가로채기 관리자에 들어가기를 시도하면 실이 감독 상태에 있는 것으로서 표시되어 있는지 검사된다. 이 표시가 있는지에 따라 실행시간 시스템은 새로운 메커니즘에 실이 적용될 수 있는지의 여부를 결정한다. 실이 가드 객체 또는 가로채기 관리자를 떠나면 지시자는 지워진다.

이 임시로 할당된 상태는, 시스템 호출을 실행하고 있는 과정이 시스템 호출을 실행하지 않는 과정보다 자원에 접근하는 권리를 더 많이 가지는 전통적인 UNIX의 시스템 호출 개념과 어느 정도 유사하다. 커널 모드에 있는 동안, 감독 상태가 실행 과정에 임시적으로 할당된다. 이 특권을 구비함으로써 과정은 운영 체제가 보유하는 보호된 자원에 대해 접근을 한다. 시스템 호출을 떠날 때, 특권 상태는 지워진다. 시스템 호출 개념의 상세한 설명은 1996년에 Addison-Wesley 출판사에 의해 간행된 레플러(S.J. Leffler) 등에 의한 '4.4 BSD UNIX Operating System'이라는 명칭의 책에 기술되어 있다.

가드 객체 및 가로채기 관리자는 또한 콘텍스트 데이터에 접근할 수 있으며 원래의 방법에 제공된 아규먼트를 포함하여 그것을 수정할 수 있다. 원래 방법의 호출이 가로채기되고 가로채기 관리자가 요청되거나 가드 객체가 호출되면, 실행 실이 원래의 방법의 스택 프레임에 대한 지시자에 임시적으로 할당된다. 따라서, 그것은 스택 상에 자리잡은 방법 아규먼트에 대해 접근한다.

본 발명에 따라 보안 요소에 들어갈 때, 적절한 가드 발송표에 대한 지시자가 실행 실에 할당된다. 따라서, 실은 모든 가드 객체에 완전하게 접근한다.

전술한 기능을 달성하기 위해, 실 구조가 다음 요소:

- 실이 감독 상태인지 아닌지를 나타내는 단일 비트(sv_thread로 표기됨)와,

- 적절한 가드 발송표에 대한 지시자(iv_filtertable로 표기됨)와,

- 원래의 방법의 적절한 스택 프레임에 대한 지시자(ic_optop으로 표기됨)

를 구비하도록 확장되는데, 이 요소는 모두 실이 보통 코드와 보안 코드 사이의 경계를 가로지를 때 할당된다.

자바 클래스 계층-사용자 레벨

이 절에서는, 자바로 구현된 요소에 대해 기술한다. 새로운 보안 메커니즘을 자바로 구현하는 것은 자바 철학의 주된 원리 중의 하나, 즉 '가능한 한 휴대할 수 있도록 하라'를 따른다.

따라서, 대부분의 구현은 원래의 C 코드에서 구현된 요소에 대한 잘 정의된 인터페이스를 이용하여 자바로 이루어진다. 원래의 C 코드로 구현된 루틴은 InterceptionInfo로 불리는 요약 기초 클래스(abstract base class)를 통해 접근될 수 있다.

가로채기 정보

InterceptionInfo는 자바로 된 요약 클래스이다. 그것은 본 발명에 따른 기본 보안 기능을 제공하는데 이용된다. 자바에서의 클래스 수정기(class modifier)에 관한 논의는 앞서 인용한 고슬린 등에 의해 제공된다. 이 기본 클래스의 요소는 다음과 같은 범주:

- 임의의 가드 객체의 설치 관리,

- 임의의 가드 객체의 제거 관리,

- 설치된 가드 객체의 순서 관리,

- 임의의 아규먼트의 인출 및 수정,

- 호출한 객체/개체의, 호출된 객체/개체의 충분히 공인된 이름 및 방법 이름 및 서명 이름의 인출

로 나뉘어 질 수 있다.

InterceptionInfo 기초 클래스에 의해 제공된 응용 프로그램 인터페이스(Application Programming Interface: API)의 개요는 코드 부분 C5에서 제공된다. 본 발명에 따른 충분한 보안 기능을 얻기 위해, 가로채기 관리자 클래스 및 모든 가드 객체는 이 InterceptionInfo 기반 클래스를 도시한 바와 같이 서브클래스화 해야 한다.

코드 부분 C5

이 클래스는 구현된 원래의 C 코드에 대한 인터페이스를 제공한다.

```
public abstract class InterceptionInfo {
  protected native void unresolveMethod();
  protected native int getIntArgument(int index)
    throws ArgumentAccessException;
  protected native void setIntArgument(int index, int value)
    throws ArgumentAccessException;
  protected native float getFloatArgument(int index)
    throws ArgumentAccessException;
  protected native void setFloatArgument(int index, float
    throws ArgumentAccessException;                    value)
  protected native byte getByteArgument(int index)
    throws ArgumentAccessException;
  protected native void setByteArgument(int index, byte
    throws ArgumentAccessException;                    value)
  protected native Object getObjectArgument(int index)
    throws ArgumentAccessException;
  protected native void setObjectArgument(int index, Object
    throws ArgumentAccessException;                    value)
  protected native void printArgument(int index);
  protected native Object getClassLoader();
  protected static native String getCaller();
  protected static native String getCallee();
  protected static native String getMethodName();
  protected static native String getSignature();
  protected static native boolean
    pushFilter(InterceptionFilter arg, String filtermethod,
      String filtersignature,
      String ic_manager_notify);
  protected static native boolean popFilter();
  protected static native Object[] getFilterList();

} // Class InterceptionInfo
```

가로채기 관리자 클래스

가로채기 관리자는 자바로 구현되고, 최종적이며 정적이라고 간주한다. 가로채기 관리자는 InterceptionInfo 기초 클래스를 서브클래스화 한다. 그것은 호출하는 개체의 이름 공간을 제어하고 수정하는 임무를 맡는다. 가로채기 관리자는, 호출 개체가 애플리케이션의 일부이든 애플릿의 일부이든 상관없이, 즉 애플리케이션과 애플릿의 구별 없이, 자바 환경에 대한 보안 정책을 구현한다. 가로채기 관리자는 보안 샌드 박스의 경계를 정의하고 샌드 박스의 외부와의 통신을 위한 게이트를 구현한다.

자바 스스로 메모리 관리를 하기 때문에, 쓰레기 수거(garbage collection)는 비사용 객체를 제거하여 메모리를 비울 책임이 있다. 객체는 자바 레벨에서 더 이상 아무런 참조도 존재하지 않는 경우에 비사용으로 간주된다. 따라서 가로채기 관리자는 그것이 생성하고 설치한 모든 가드 객체를 추적 감시해야 하며, 그렇게 하지 않으면 메모리를 비우는 동안에 쓰레기 수거가 이 객체를 제거해 버릴 것이다. 쓰레기 수거에 대한 상세한 논의는 앞서 인용한 린드홀름의 책에서 제공된다. 가로채기 관리자는 가드 객체를 생성하고, 어느 가드 객체를 어느 방법에 부착해야 하는지를 결정한다. 관리자는 또한 가드 객체들이 원래의 방법 전 및/또는 후에 호출될 순서를 지정한다.

가로채기 필터

InterceptionFilter 클래스는 가드 객체가 요구하는 기본 기능을 제공한다. 그것은 가로채기 관리자와 같은 원래의 C 코드 방법에 접근하기 위해 InterceptionFilter 클래스를 서브클래스화 한다. 이 클래스는 서브클래스화된 것을 원래의 방법(그것이 부착된 객체 참조)으로부터 그 자체를 분리하기 위한 방법으로 확장한다. 실제 구현에서는, 가드 객체는 원래의 방법이 호출되기 전에 실행된다. 코드 부분 C6은 기본적인 가드 객체 기초 클래스를 보여준다. defaultCallFilter() 방법은 객체 참조가 호출되기 전에 디폴트로 호출되는 방법이다.

코드 부분 C6

이것은 InterceptionInfo 요약 기초 클래스를 확장한다.

```
public class InterceptionFilter extends InterceptionInfo {
   protected int unresolve;
   private protected InterceptionFilter() {
      unresolve = 0;
   }
   private protected void defaultCallFilter() {
   throw new ICFilterException("no filter method
   implemented");
   }
   private protected native void detachMySelf();
}
```

결론

이 장 B는 장 A에서 표현된 개념의 구현이라는 핵심 주제를 다루었다. 본 발명의 보안 메커니즘을 구현하는 본질적인 기능은 제안된 아키텍쳐의 부분 집합으로서 서술되었는데, 이 아키텍쳐는 시스템 자원을 나타내는 시스템 클래스에 대한 접근을 제어하고 임의의 가드 객체를 호출하기 위한 원리를 말한다.

자바 가상 머신에서 수정은 최소한 한정되었다. 호출한 객체의 확정 이름 공간을 수정하기 위해 이름 해결 과정에서 중대한 변화가 이루어졌고, 가드 객체를 호출하기 위한 방법 호출 과정에서도 중대한 변화가 있었다. 가드 객체를 통하여, 혹은 양자택일적으로 그리고 선택적으로 이름 해결 과정 중에 오직 한번 함으로써 이제 모든 방법 호출에 대해 보안성은 보장될 수 있다. 가로채기 관리자와 가드 객체 모두로 인해, 프로그래머는 경제적 관점에서뿐만 아니라 원하는 보호 수준의 측면에서도 최적화될 수 있는 가변 결정립 보호 메커니즘을 구성할 수 있다.

(57) 청구의 범위

청구항 1

이름 해결 메커니즘(name resolution mechanism)을 구비하는 데이터 프로세싱 시스템, 특히 분산 네트워크 시스템에서의, 예를 들어 파일 또는 기타 객체와 같은 자원(2a,...,2n)을 무단 접근으로부터 보호하기 위한 자원 보호 방법에 있어서, 상기 자원은 제 1 개체(entity), 즉 호출하는 개체(7)에 의해 호출되고 상기 시스템 내의 제 2 개체, 즉 호출되는 개체(9)에 의해 제공된 것이며,

상기 자원에 대한 접근은 상징 이름(symbolic name)을 이용하여 상기 호출하는 개체(7)에 의해 호출되는데, 여기서 상징 이름 전체는 개체에 대한 추정 이름 공간(presumed name space)을 형성하며,

상기 이름 해결 과정은 상기 상징 이름에 대해 객체 참조를 할당하되, 여기서 객체 참조 전체는 허가된 자원을 반영하는 개체에 대한 확정 이름 공간(concrete name space)을 형성하고, 상기 허가된 자원은 상기 시스템의 모든 자원의 부분 집합이며,

상기 호출하는 개체(7)는 상기 호출된 개체(9) 내에서 상기 확정 이름 공간에 대해서만 접근을 허가 받는

분산 네트워크 시스템에서의 자원 보호 방법.

청구항 2

제 1 항에 있어서,

상기 상징 이름은 적어도 개체 이름과 방법 이름을 포함하는데, 상기 이름 전체는 상기 추정 이름 공간을 형성하는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 3

제 1 항 내지 2 항 중 어느 한 항에 있어서,

상기 접근이 허용되면, 상기 이름 해결 과정은 호출된 개체(9)에서 원하는 방법을 호출하여 원하는 자원(2a,...,2n)에 접근함으로써, 호출하는 개체(7)의 접근, 특히 즉각적인 접근을 허가하는 객체 참조(6)로 상징 이름을 해결하는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 4

제 1 항 내지 3 항 중 어느 한 항에 있어서,

가드 객체(13)를 제공하고 상기 객체 참조(6)를 상기 가드 객체에 대한 참조로 대체함으로써 상기 문제 해결 과정을 수정하는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 5

제 4 항에 있어서,

상기 가드 객체가 상기 객체 참조(6b)를 상기 호출 개체(7)로부터 감춤으로써 원하는 방법에 대한 조건부 접근을 제공하고 상기 호출 과정을 제어하는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 6

제 4 항 또는 5 항에 있어서,

가드 객체는 객체 참조가 사용되기 전 또는 후에 호출되는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 7

제 4 항 또는 제 5 항에 있어서,

상기 가드 객체가 제공한 조건부 접근이, 미세 파손시 구현되는, 즉 이름 해결 과정에서 사용된 파손에 비해 더 상세한 분석에 의존하는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 8

제 1 항 내지 제 7 항 중의 어느 한 항에 있어서,

상기 방법의 적어도 하나의 단계가 객체 지향 방식으로 구현되는 분산 네트워크 시스템에서의 자원 보호 방법.

청구항 9

데이터 처리 네트워크에서의, 예를 들어 파일 또는 기타 객체와 같은 자원(2a,...,2n)을 무단 접근으로부터 보호하기 위한 수단을 포함하는 자원 보호를 위한 접근 제어 장치에 있어서, 상기 자원은 제 1 개체, 즉 호출하는 개체(7)에 의해 호출되고 상기 네트워크 내의 제 2 개체, 즉 호출되는 개체(9)에 의해 제공된 상기 접근 제어 장치가,

상기 호출하는 개체(7)가 상기 자원에 대한 접근을 호출하는데 사용하는 상징 이름을 해결하되, 상기 상징 이름 전체는 개체에 대한 추정 이름 공간을 형성하는 상징 이름 해결 수단과,

상기 각 상징 이름에 대해 객체 참조를 할당하되, 상기 객체 참조 전체는 허가된 자원을 반영하는 확정 이름 공간을 형성하고, 상기 허가된 자원은 모든 자원의 부분 집합인 객체 참조 할당 수단과,

상기 호출되는 개체(9) 내에서 상기 확정 이름 공간에 대한 상기 호출하는 개체(7)의 접근을 제한하는 접근 제한 수단

을 포함하는 데이터 처리 네트워크에서의 접근 제어 장치.

청구항 10

제 9 항에 있어서,

상기 장치가 가드 객체(13)를 설치하고 상기 객체 참조(6)를 상기 가드 객체에 대한 참조로 대체하는 수단을 더 포함하는 데이터 처리 네트워크에서의 접근 제어 장치.

청구항 11

제 9 항 또는 제 10 항에 있어서,

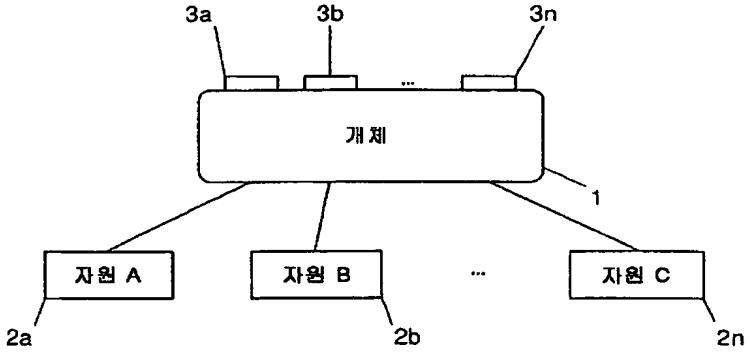
상기 수단들 중의 적어도 하나는 객체 지향 방식의 소프트웨어로 구현되는 데이터 처리 네트워크에서의 접근 제어 장치.

청구항 12

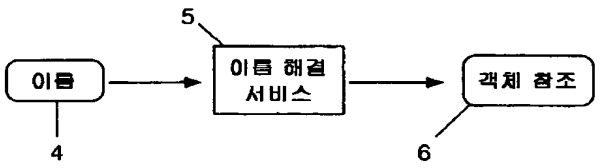
제 1 항 내지 제 8 항 중의 어느 한 항을 따르는 방법 및/또는 제 9 항 내지 제 11 항 중의 어느 한 항을 따르는 이름 해결 시스템이 구현되는 데이터 처리 네트워크.

도면

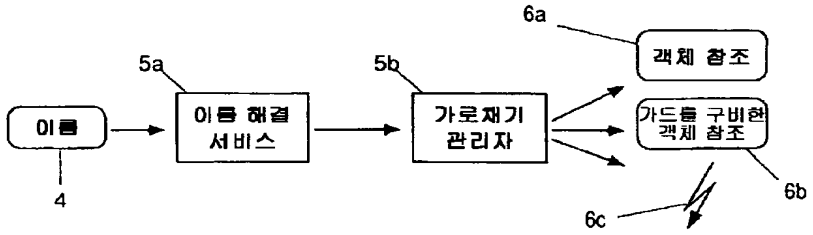
도면1



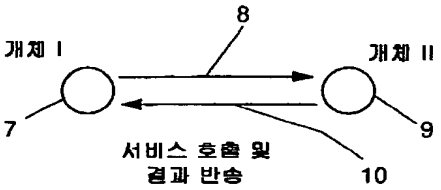
도면2a



도면2b

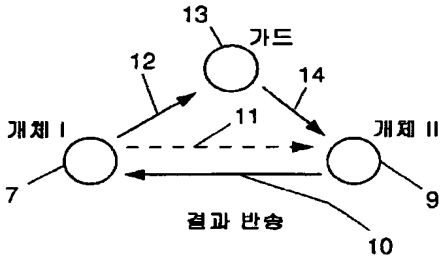


도면3a

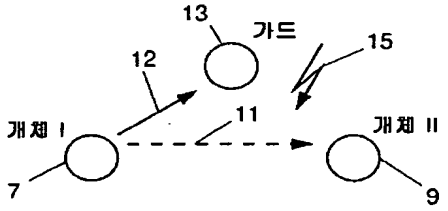


16-15

도면3b



도면3c



도면4